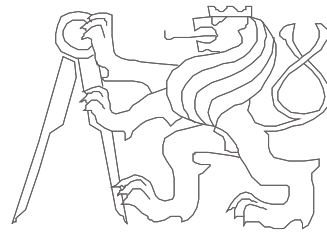


Architektura počítačů

Multiprocesorové systémy, problém koherence

Michal Štepanovský



České vysoké učení technické v Praze

Ver.1.00

- Klasifikace multiprocessorových systémů
- Pojem koherence a její definice
- Koherenční protokoly (snooping-based):
 - WTWNA
 - WBWA
 - MESI
- Koherenční protokoly pro rozsáhlé systémy (directory-based)

Multiprocessorové systémy

Terminologický posun: Multiprocessorové systémy = systémy s více procesory. Ale mohou to být i dnešní procesory „jen“ s více jádry.

Softwarový pohled (jak to vidí programátor):

- Systémy **se sdílenou pamětí** (shared memory system - **SMS**). Běží jedna instance OS (společné plánování), Standard: OpenMP, taktéž lze použít MPI (Message Passing Interface).
 - Výhody: snazší programování
- Systémy **s oddělenou pamětí** (distributed memory system - **DMS**): komunikují pomocí předávání zpráv – message passing. Na každém uzlu (procesoru, skupině procesorů - hybridní) jiná instance OS. Síťové protokoly, RPC, Standard: MPI. Někdy se označují zkratkou NoRMA (No Remote Memory Access)
 - Výhody: méně potřebného HW, snazší škálování.

Hardwarový pohled:

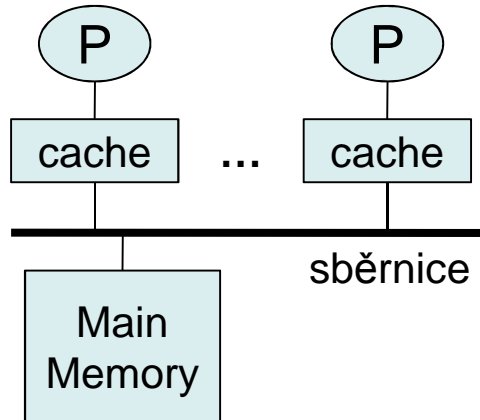
- Systémy se **sdílenou fyzickou pamětí** – společný fyzický adresní prostor – SMP: UMA
- Systémy s **oddělenou fyzickou pamětí** – každý uzel má nezávislý fyzický adresní prostor – a ten může být společný (NUMA) nebo privátní (cluster)

Vymezení pojmů:

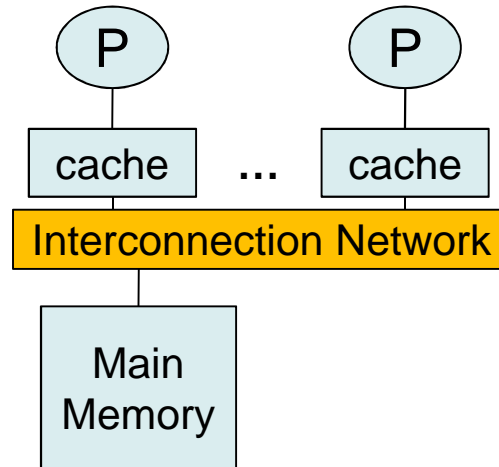
- **SMP** – Symetrický multiprocessor – procesory jsou připojeny k centrální společné paměti. Procesory jsou **totožné** a „vidí“ paměť i zbytek systému stejně.
- **UMA** – Uniform Memory Access – ze všech procesorů trvá přístup do paměti stejně dlouho. Systémy UMA jsou dnes nejčastěji implementovány právě jako SMP.
- **NUMA** – Non-Uniform Memory Access – čas přístupu do paměti záleží od místa (adresy) kam přistupujeme. Do vlastní paměti rychleji, do vzdálené pomaleji.
- **ccNUMA** – cache-coherent NUMA – koherence je zabezpečena na HW úrovni
- **Cluster** – skupina spolupracujících počítačů propojených rychlou sítí, na které můžeme nahlížet jako na jeden výpočetní systém

Multiprocessorové systémy - příklady

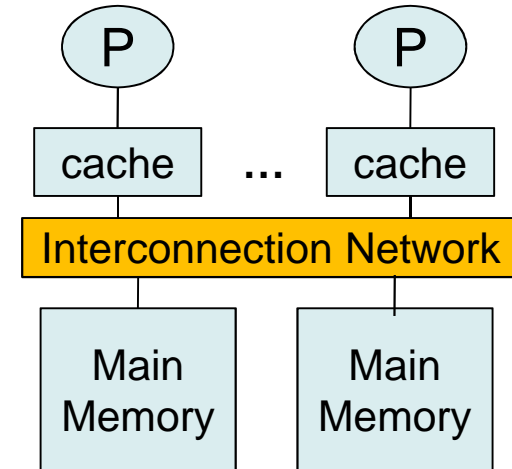
Tradiční SMP, zároveň UMA



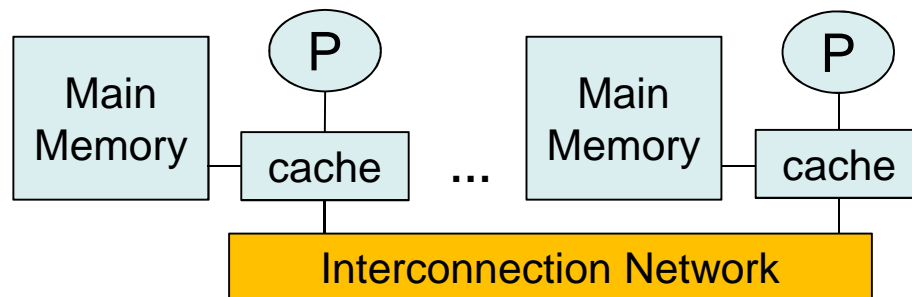
UMA



UMA



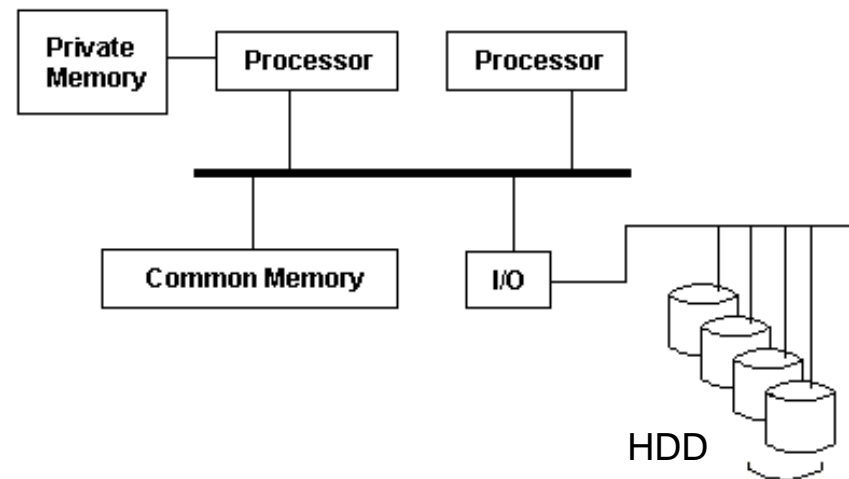
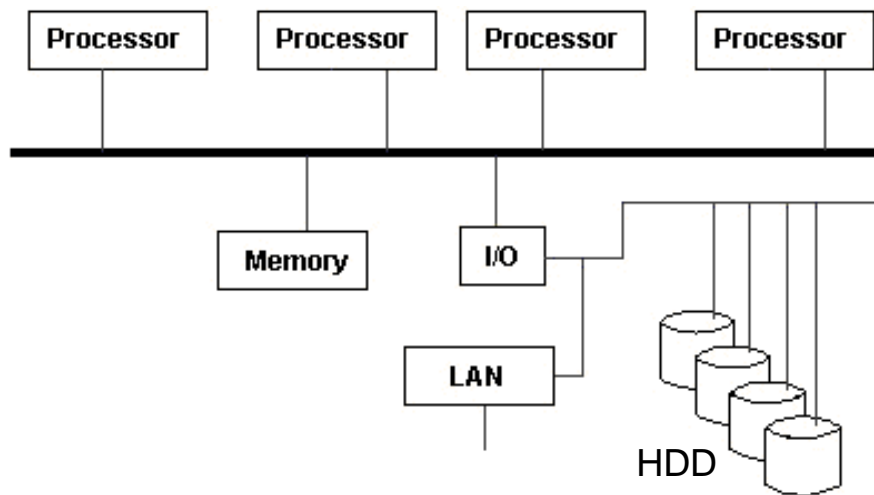
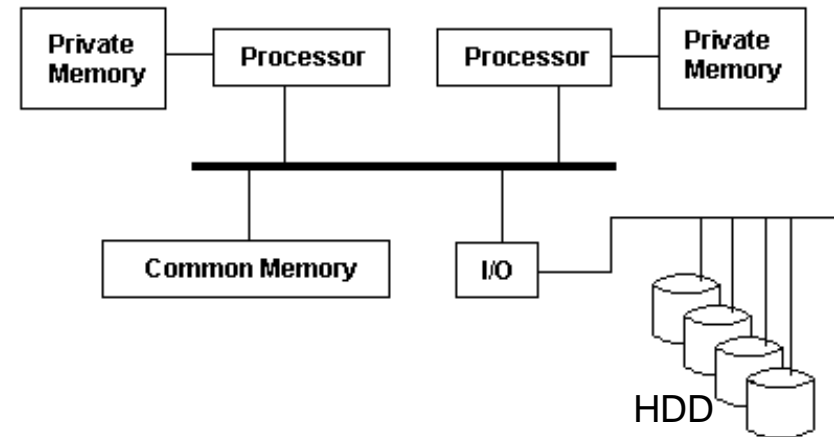
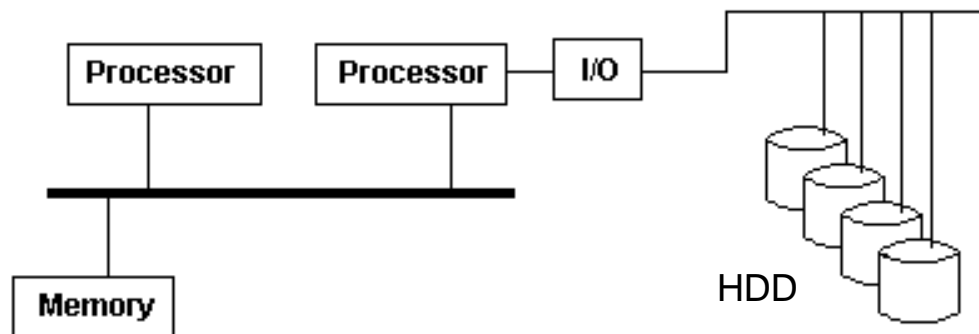
NUMA



Interconnection Network

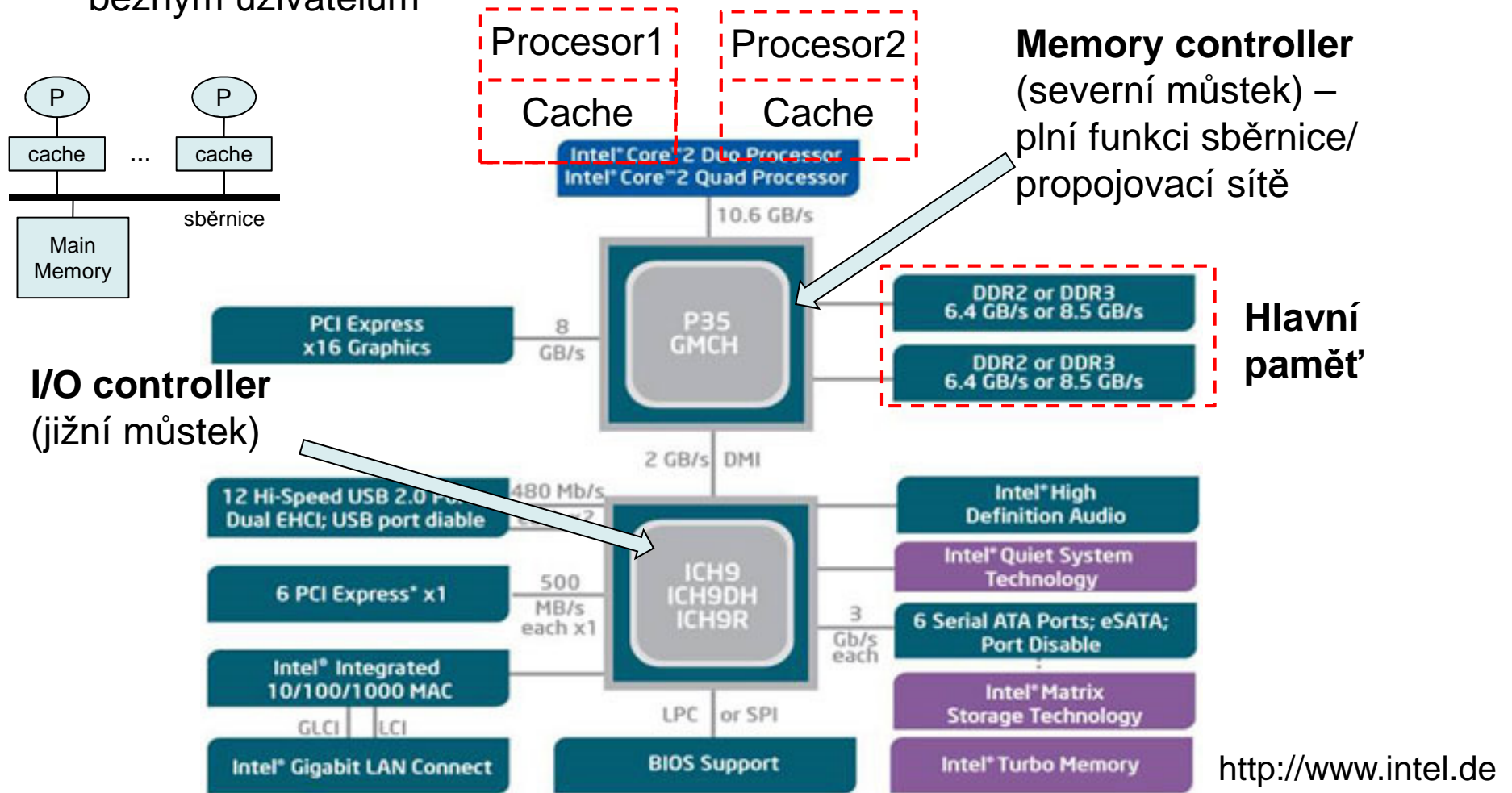
Může se jednat o křížový přepínač (crossbar), nebo dynamickou vícestupňovou síť, nebo o nějaký typ statické propojovací sítě

Najdete SMP ???



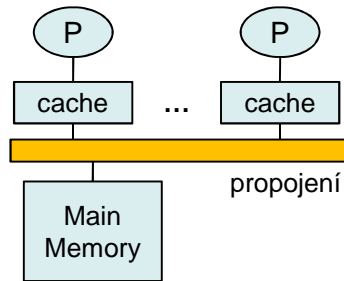
Historie a dnes

- Multiprocessorové systémy existují již desítky let a byly využívány k náročným výpočtům ve vědě a v komerci...
- S příchodem prvních více-jádrových procesorů se však model SMP přiblížil i běžným uživatelům

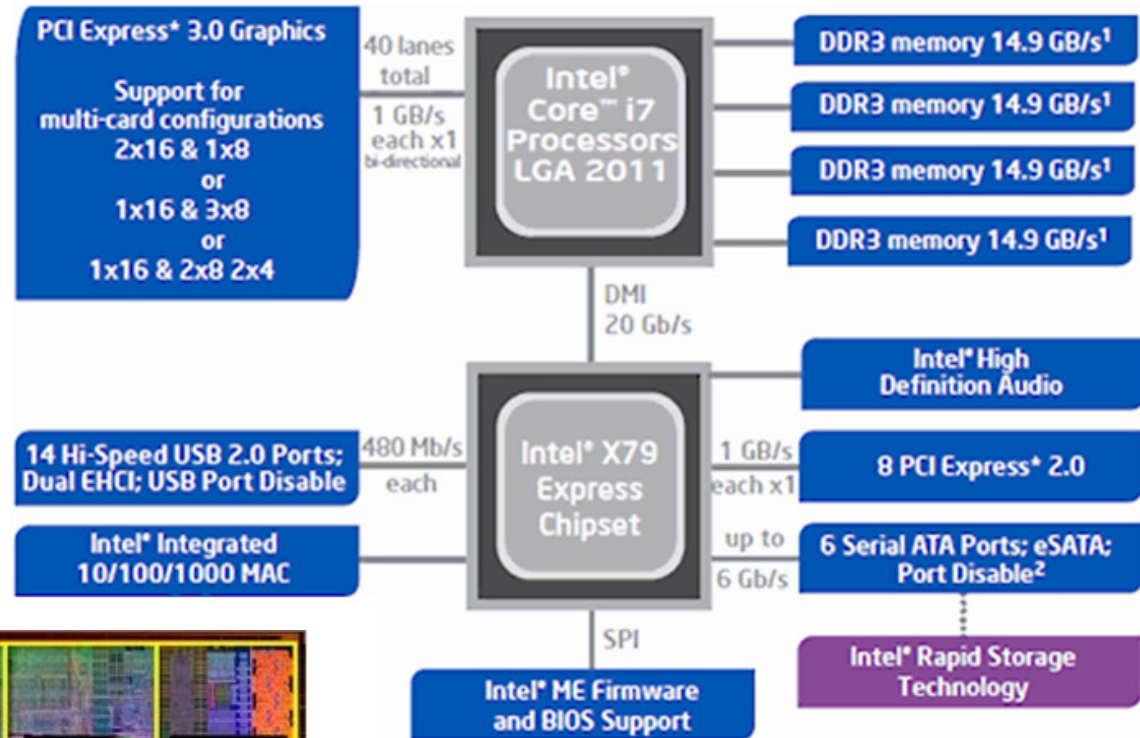
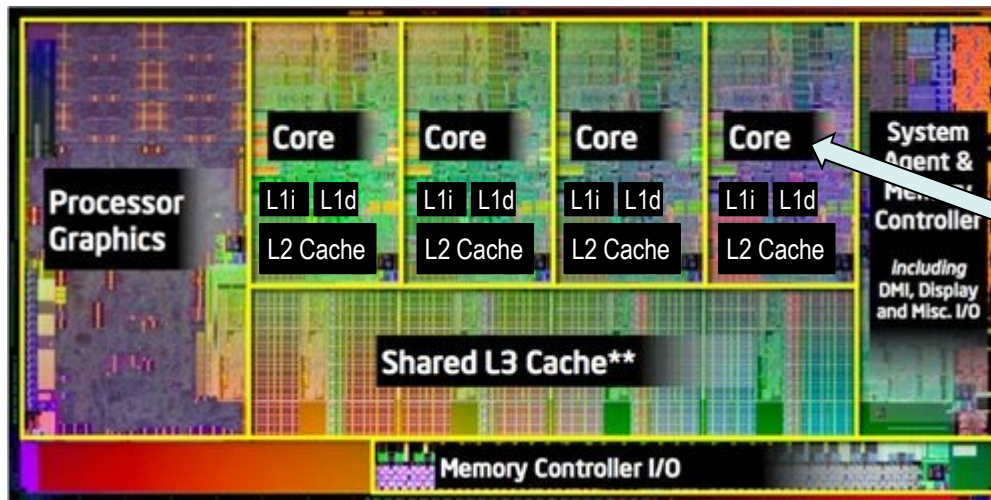


Historie a dnes

- Další vývoj posunul funkci severního můstku přímo do procesoru. Řadič paměti tedy najdeme tam.



Core i7-2600K

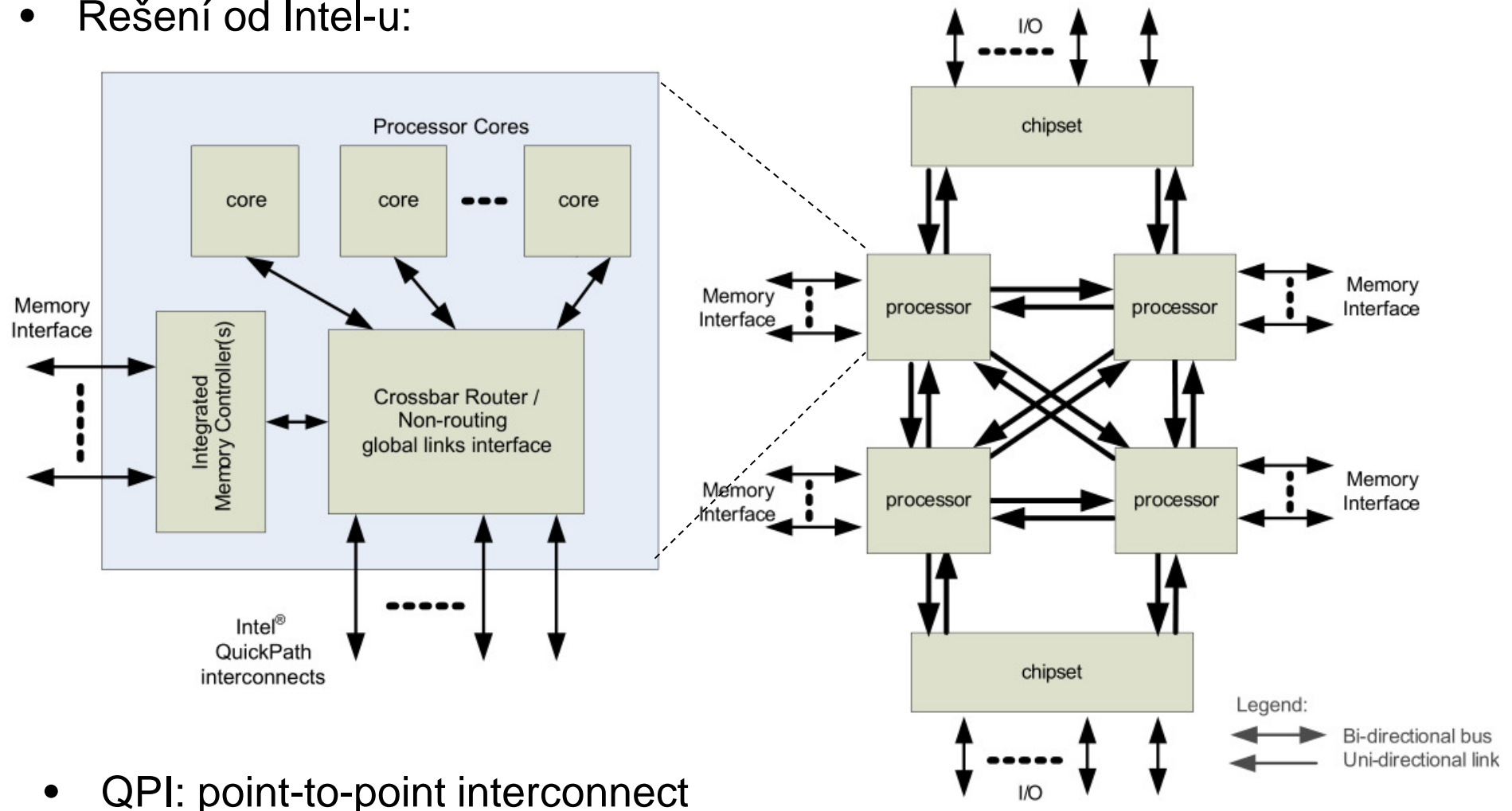


<http://www.intel.de>

Na core můžeme nahlížet jako na procesor. Pokud je L3 cache inkuzivní, pak ji z pohledu koherence můžeme chápat jako vnější paměť.

Historie a dnes

- Pomocí QuickPath Interconnect pak můžeme navzájem propojit několik procesorů.. Tím se dostáváme k systému **NUMA**.
- Řešení od Intel-u:



- QPI: point-to-point interconnect

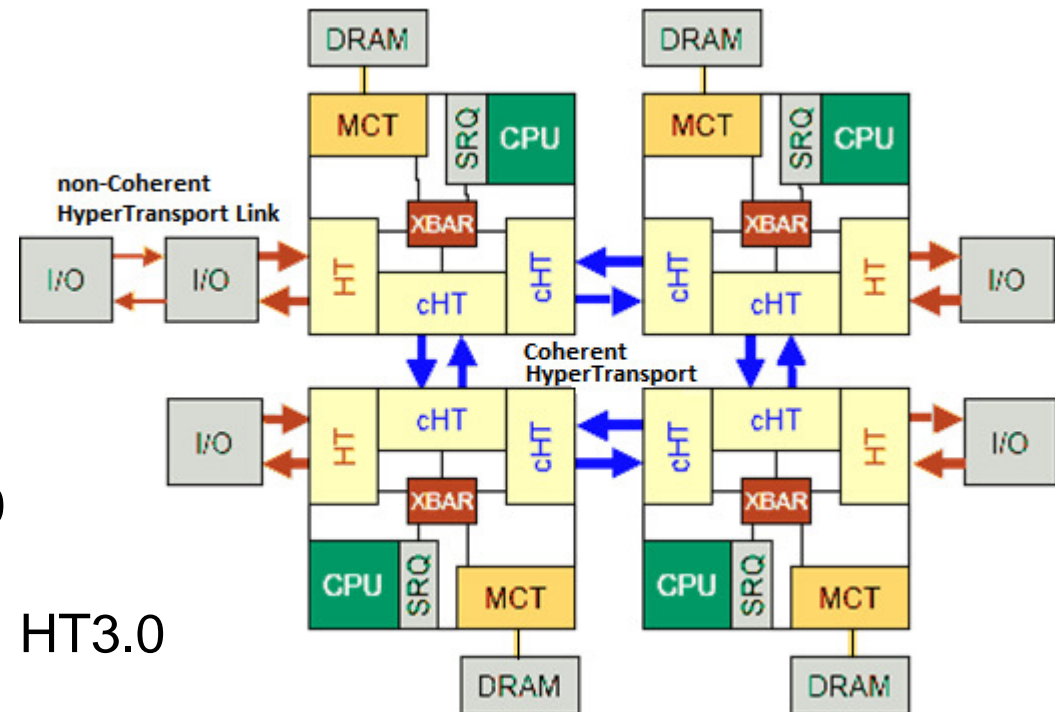
Historie a dnes

- Řešení od AMD (HT - HyperTransport):

Motherboard



- **Čtyři AMD Opteron™ 6000** series processor (Socket G34) **16/12/8/4-Core ready**; HT3.0 Link support
- **Up to 1TB DDR3 1600MHz**
- 6x SATA2
- 1x PCI-E 2.0 x16 slot

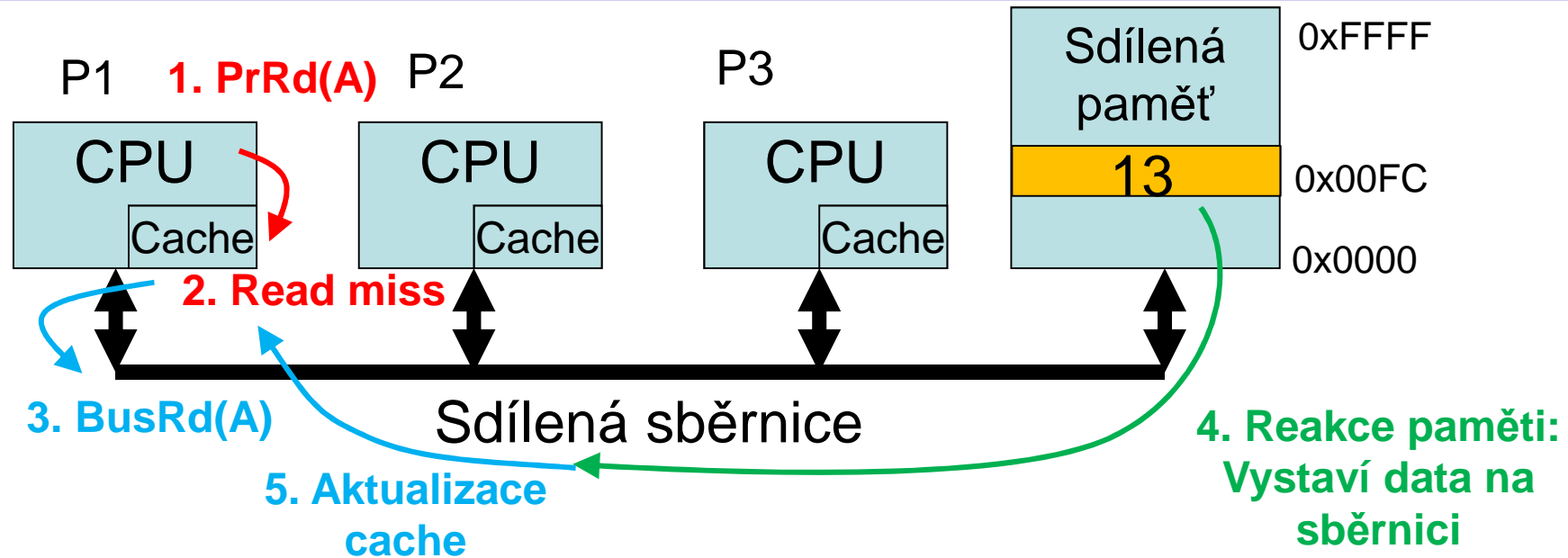


Hlavní téma dnešní přednášky:

Systemy se sdílenou pamětí

- Přirozené rozšíření jednoprocessorových počítačů přidáním dalších procesorů (či jader).
- Tradiční jednoprocessorové OS jsou modifikovatelné. Použijí se k plánování procesů pro více procesorů.
- Víceprocesový/vícevláknový program poběží na systémech s jedním procesem v režimu sdílení času (Timesharing). Využije ale i více procesorů, pokud jsou k dispozici.
- Vhodný model pro úlohy s převažujícím sdílením dat, ta se sdílejí automaticky. Toto řeší transparentně HW.
- Obtížně se ale škáluje (pro větší počty procesorů).

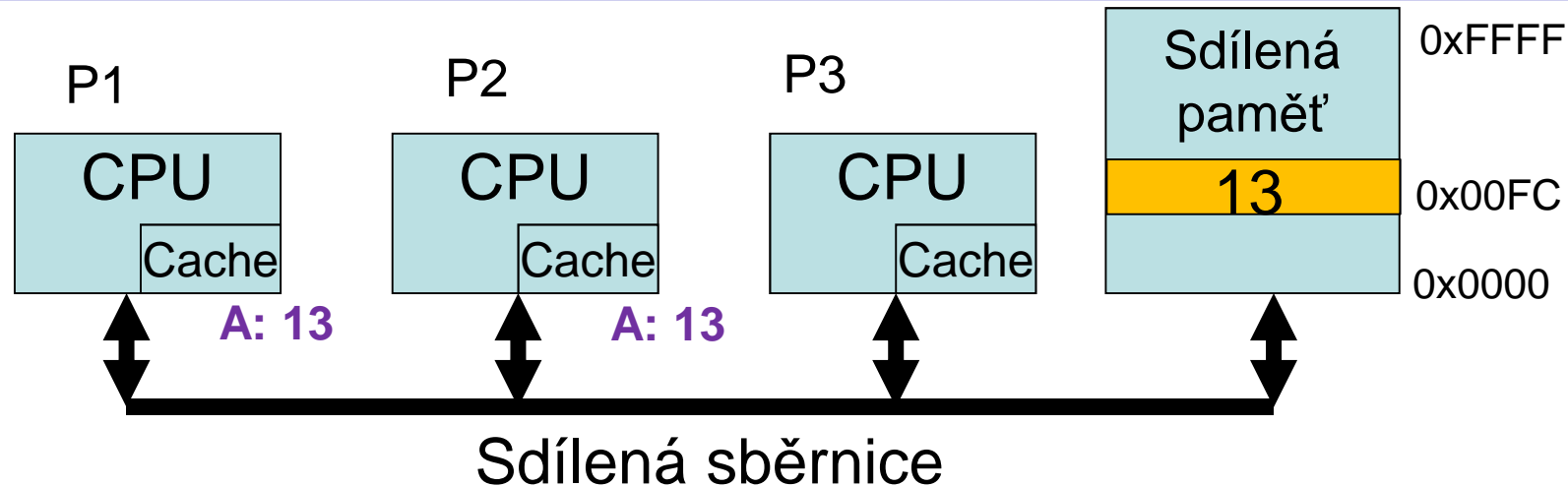
V čem je základní problém? Uvažujme write-back cache



Procesor P1 chce načíst data z adresy A (0x00FC). Co se stane?

1. Procesor P1 vyšle dotaz PrRd(A) do své cache.
2. Data v cache nenajde => cache read miss
3. Řadič cache procesoru P1 generuje BusRd(A) na sběrnici.
4. Řadič paměti vidí dotaz na čtení z konkrétní adresy, proto poskytne data z této adresy (0x00FC) s aktuální hodnotou 13.
5. Řadič cache procesoru P1 data se sběrnice vyčte a uloží do své cache.

V čem je základní problém? Uvažujme write-back cache



Procesor P1 chce načíst data z adresy A (0x00FC). Co se stane?

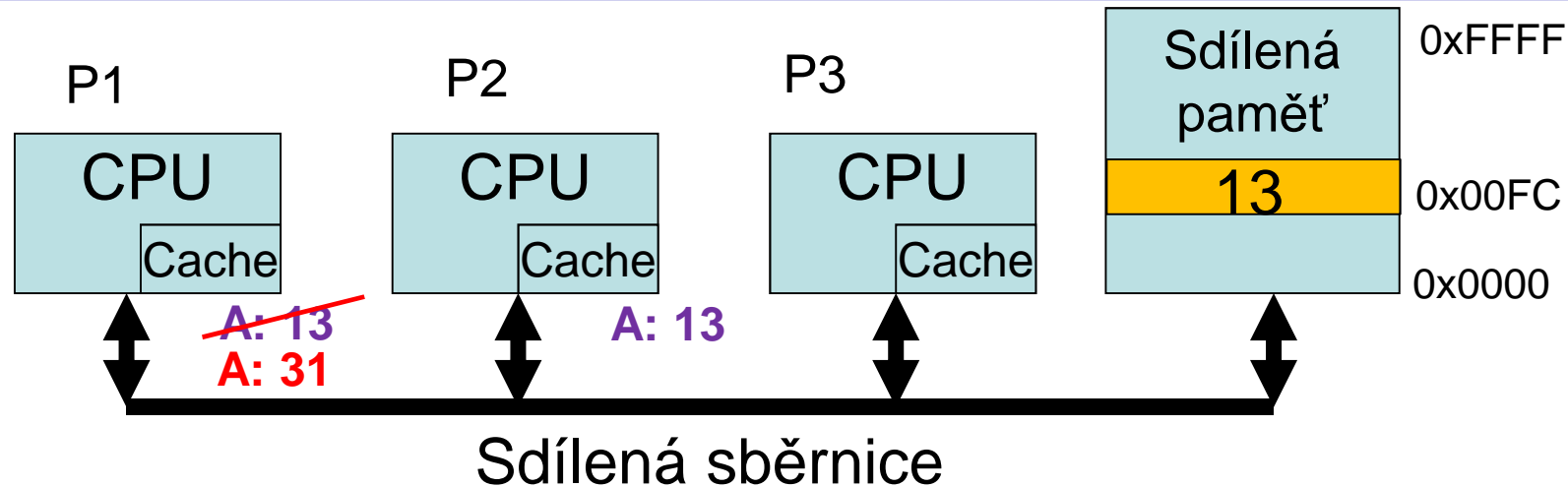
- P1 získal data do své cache. A: 13

Procesor P2 chce načíst data z adresy A (0x00FC). Co se stane?

- To samé. P2 získá data z paměti a uloží do cache. A: 13

Nyní mohou oba procesory opakovaně nezávisle číst položku A.

V čem je základní problém? Uvažujme write-back cache



Procesor P1 chce načíst data z adresy A (0x00FC). Co se stane?

- P1 získal data do své cache. A: 13

Procesor P2 chce načíst data z adresy A (0x00FC). Co se stane?

- To samé. P2 získá data z paměti a uloží do cache. A: 13

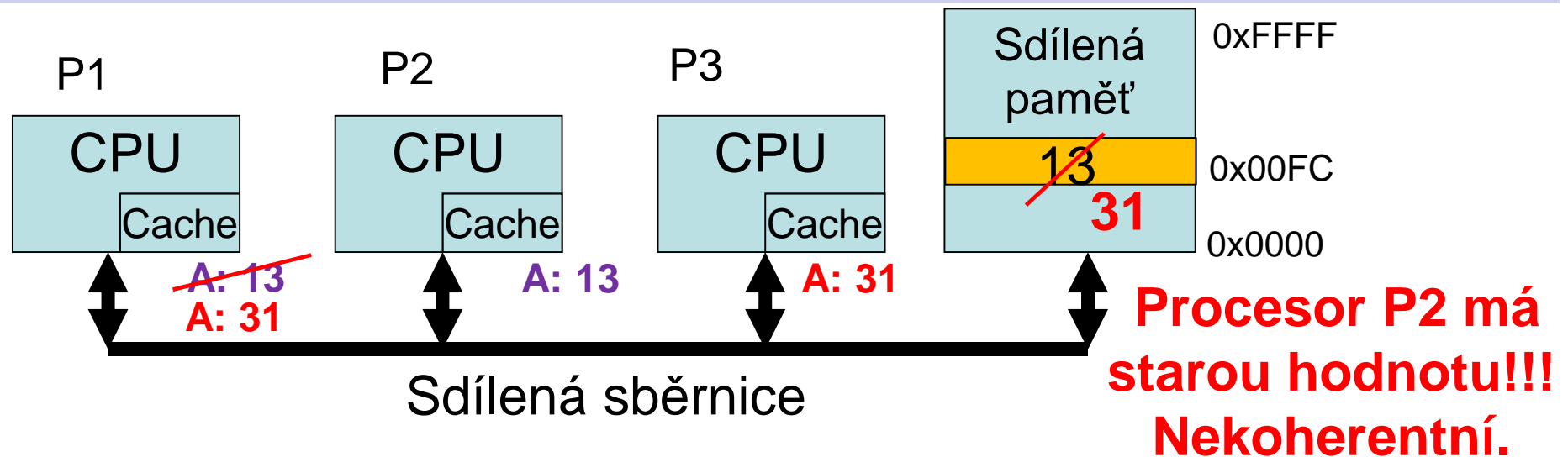
Procesor P1 zapíše novou hodnotu do A. Například 31.

- Hodnota v jeho cache je modifikována.

Procesor P3 chce načíst data z adresy A. Jakou hodnotu načte?

- Paměť vydá hodnotu 13. Procesor P1 ale pracuje s 31. > **Nekoherentní**

V čem je základní problém? Uvažujme **write-through** cache



Processor P1 chce načíst data z adresy A (0x00FC). Co se stane?

- P1 získal data do své cache. A: 13

Processor P2 chce načíst data z adresy A (0x00FC). Co se stane?

- To samé. P2 získá data z paměti a uloží do cache. A: 13

Processor P1 zapíše novou hodnotu do A. Například 31.

- Hodnota v jeho cache je modifikována **a propagována do paměti.**

Processor P3 chce načíst data z adresy A. Jakou hodnotu načte?

- Paměť vydá hodnotu **31**. Processor P1 ale pracuje s **31**. **> koherentní?**

Co jsme zjistili?

- Problém spočívá v paměťové **nekoherenci**:
- Procesor data (správně) modifikoval ve své skryté paměti.
- Změna obsahu hlavní paměti nestačí.
- Skryté paměti ostatních procesorů mohou obsahovat neaktuální data.

Důležité pojmy:

- Paměťová **koherence** => **dnešní přednáška**
 - Pravidla pro přístupy k jednotlivým paměťovým místům.
- Paměťová **konzistence** => **příští přednáška**
 - Pravidla pro všechny paměťové operace v paralelním počítači. Sekvenční konzistence: “Počítač je **sekvenčně konzistentní**, jestliže je výsledek provádění programu stejný, jako kdyby operace na všech procesorech byly provedeny v nějakém sekvenčním pořadí a operace každého jednotlivého procesoru se objevují v této posloupnosti v pořadí daném jejich programem.”

- Definice: Řekneme že multiprocessorový paměťový systém je **koherentní** jestliže
 - výsledek jakéhokoli provádění programu je takový, že pro každé paměťové místo je možné sestavit myšlené sériové pořadí čtení a zápisů k tomuto paměťovému místu a platí:
 - 1. Paměťové operace k danému paměťovému místu pro každý proces se provedou v pořadí, ve kterém byly tímto procesem spuštěny.
 - 2. Hodnoty vrácené každou operací čtení jsou hodnotami naposledy provedené operace zápis do daného paměťového místa s ohledem na sériové pořadí.
- Metody pro zajištění koherence nazýváme **koherenční protokoly**.

1. Snooping

- Snooping = slídění, špehování.
- Vyžaduje doplnění cache o HW, který sleduje dění na **sběrnici** a
 - Detekuje relevantní transakce,
 - Aktualizuje stavy relevantních bloků dat,
 - Generuje paměťové transakce
- reálně používané protokoly jsou MESI, MSI, MEI, MOESI, MESIF a jejich varianty.

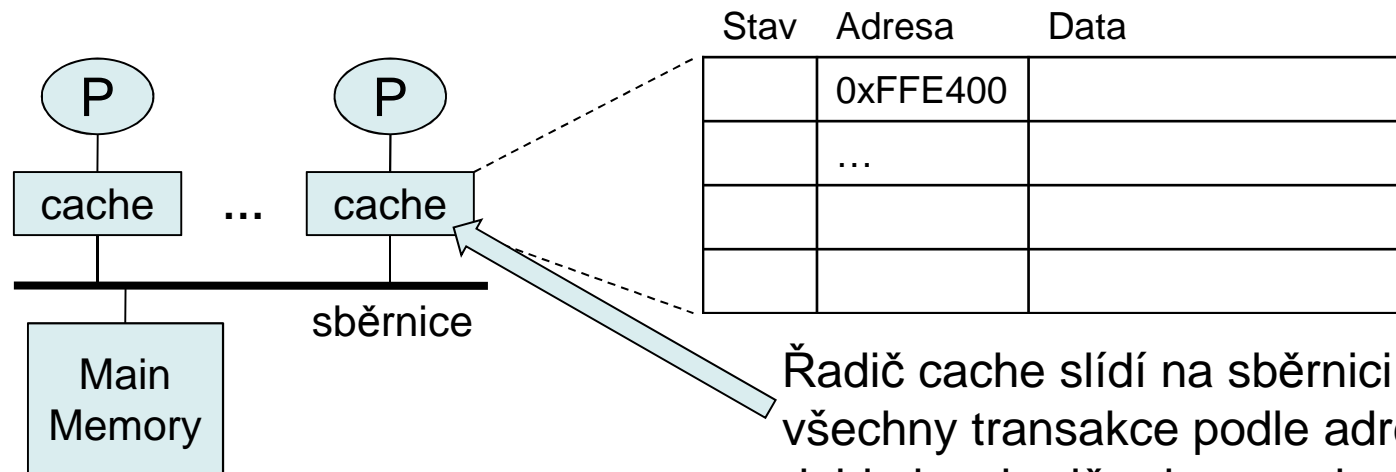
2. Directory based

- Pro rozsáhlé systémy když není k dispozici sdílená sběrnice

Bus Snooping

Varianta, kdy každý procesor ví, kdo má kopii jeho v cache uložených dat, je příliš složitá. Proto:

- Řadič každé cache spojitě slídí (snoops) na sběrnici **po zápisech** týkajících se dat **na adresách, které obsahuje**.
- To vyžaduje, aby byla struktura sběrnice globální, aby ji viděli všichni.
- Právě zde je problém ve škálovatelnosti.
- Lépe škálovatelné řešení je právě „Directory based“.



Řadič cache slídí na sběrnici a sleduje všechny transakce podle adresy. Pokud dojde ke shodě adres s adresou v dané cache, patřičně zareaguje.

Snoopovací protokoly: **zápis s aktualizací**, **zápis se zneplatněním**

- **Zápis s aktualizací**

- Procesor, který zapisuje, musí nejprve sběrnici získat. Pak na ní vyše nová data. (Všechny procesory vč. paměti jsou připojeny na tu samou sdílenou sběrnici.)
- **Všechny úspěšně slídící řadiče cache aktualizují svůj obsah (i paměť).**
- Značně zatěžuje sběrnici. **Nepoužívá se.**

- **Zápis se zneplatněním**

- Procesor zapisuje na adresu. To musí vyvolat zprávu o provedení zneplatnění obsahu na všech místech, které stejnou kopii obsahují.
- Všechny slídící řadiče cache-í provedou zneplatnění ve svém obsahu.
- Tím je zabezpečeno, že kopie řádky v cache zapisujícího procesoru je jediná platná kopie v systému. Takže procesor může libovolně zapisovat do řádky cache bez zatěžování sběrnice (strategie *Write allocate*)..
- Všechna možná čtení v ostatních procesorech teď narazí na výpadek v cache a musí data načíst.
- **Invalidační protokoly** potřebují rozlišit **alespoň dva stavy řádky cache – platná (modifikovaná), neplatná.**

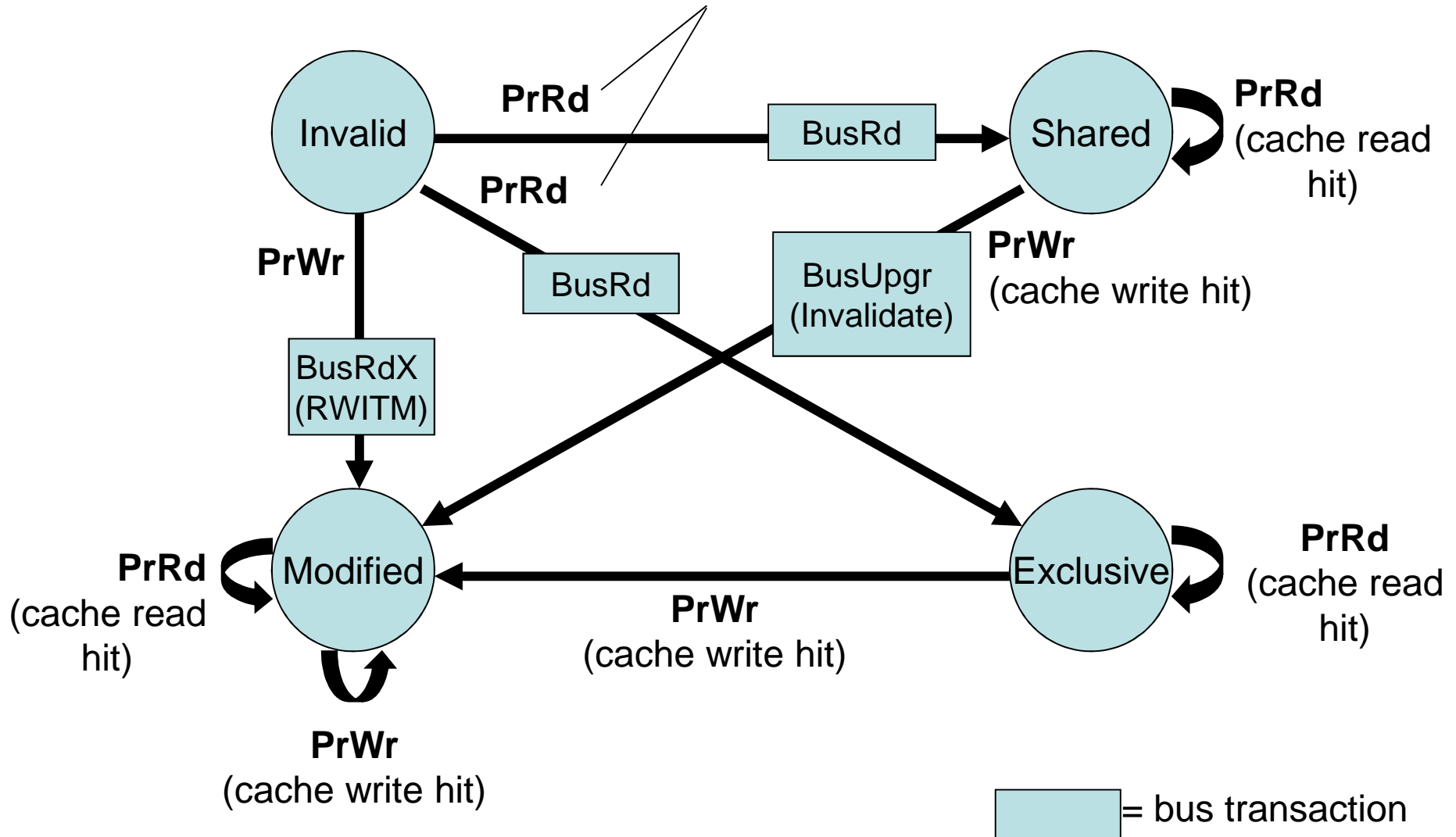
MESI Protokol

- Jde o protokol, který při zneplatnění hodnoty v multiprocessorovém systému minimalizuje akce sběrnice.
- Je založen na zpětném zápisu, tedy se v cache neaktualizují špinavé řádky dokud nehrozí, že o blok ve skryté paměti přijdeme.
- Vyžaduje rozšíření příznaků v cache (tags). Zatím tam byly jen příznaky zneplatnění (Invalid) a špinavý (Dirty).
- **Každý řádek cache může být v jednom z těchto 4 stavů (kóduje se 2 bity)**
 - **M** – Modified. Obsah řádku v cache se liší od obsahu paměti, (jinak odpovídá běžnému Dirty),
 - **E** – Exclusive. Je jedině v této cache a je stejný, jako v paměti.
 - **S** – Shared. Je stejný, jako v paměti, ale nemusí být jen v této SP.
 - **I** – Invalid. Obsah řádku neplatí.

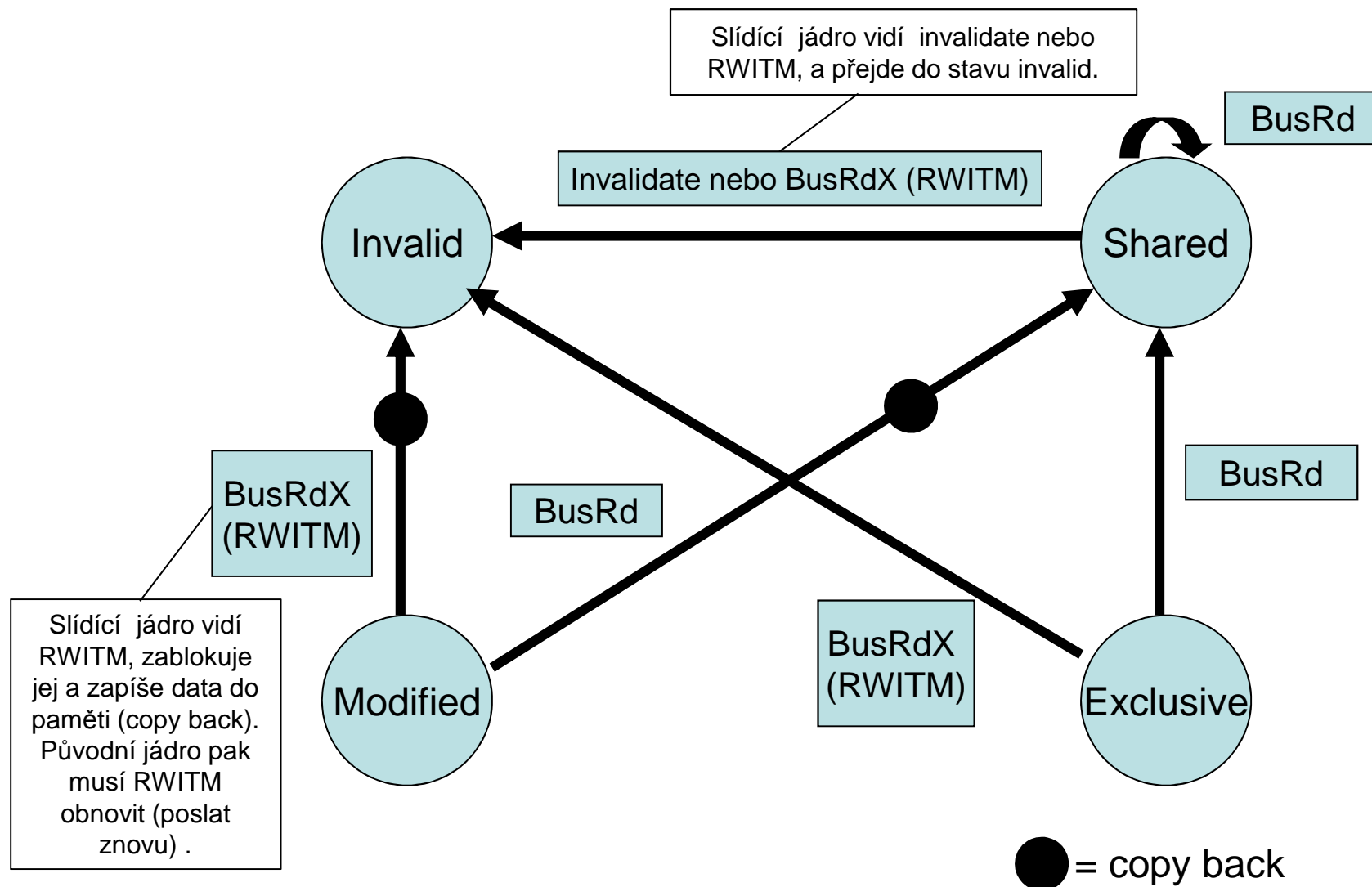
- Kompaktně popsané akce shrnuje **stavový diagram přechodů**. Ten zobrazuje, co se stane s řádkou v cache daného procesoru v případě
 - Přístupu tohoto procesoru k paměti (read hit/miss, write hit/miss)
 - Přístupu k paměti jiným procesorem, který úspěšně vyslídí tento řadič cache (Mem read, RWITM = Read With Intent To Modify, Invalidate).
- Akce připojeného/lokálního procesoru:
 - Read Hit (čtená hodnota je k dispozici)
 - Read Miss (výpadek při čtení)
 - Write Hit (zápis byl úspěšný)
 - Write Miss (výpadek při zápisu)

MESI – lokální procesor

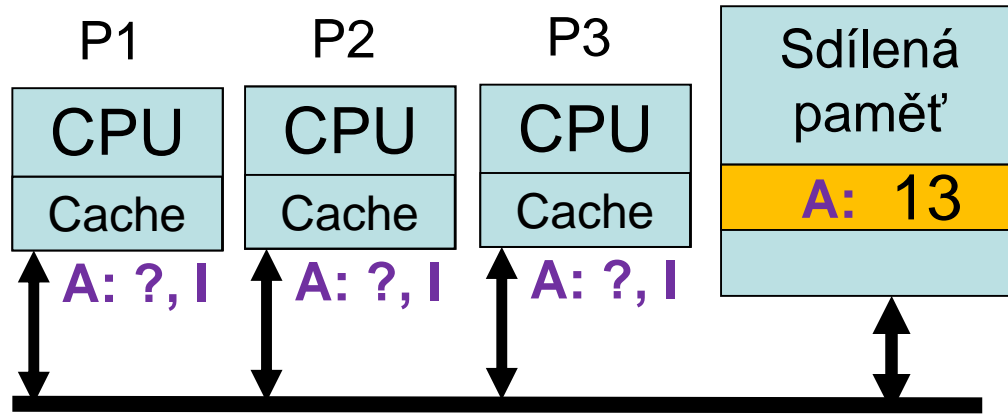
Po které hraně se se půjde se rozhodne po uplynutí slídící doby (snoop done) podle speciálního vodiče sběrnice signalizujícího, zda existuje sdílená kopie.



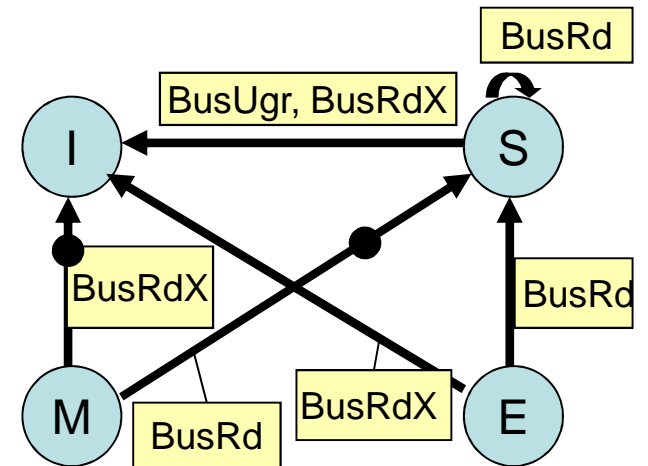
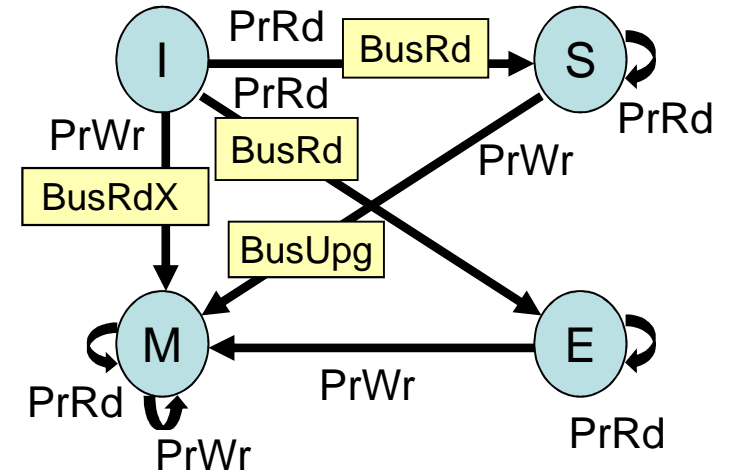
MESI – slídící procesor



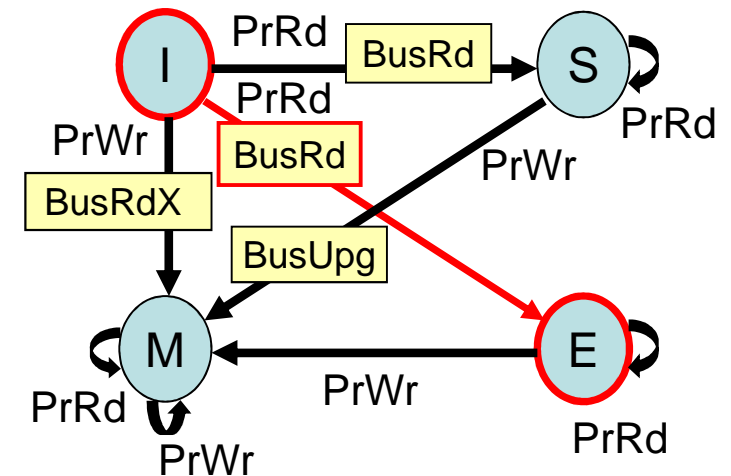
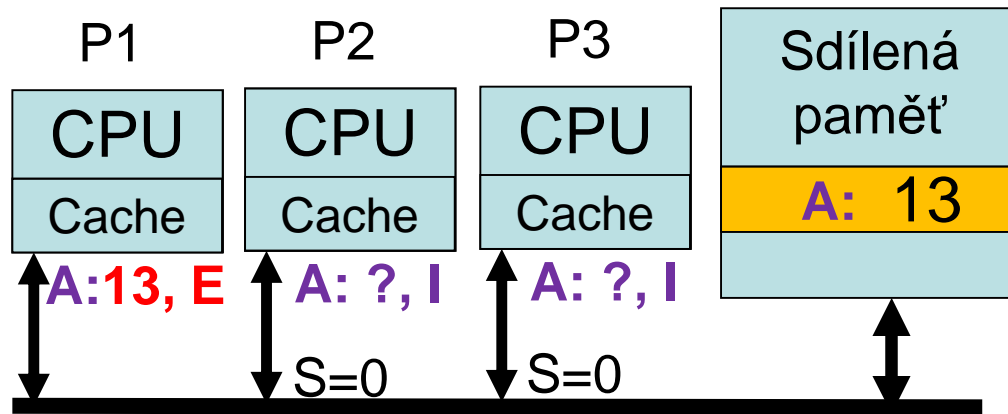
Příklad – Protokol MESI



Počáteční stav.

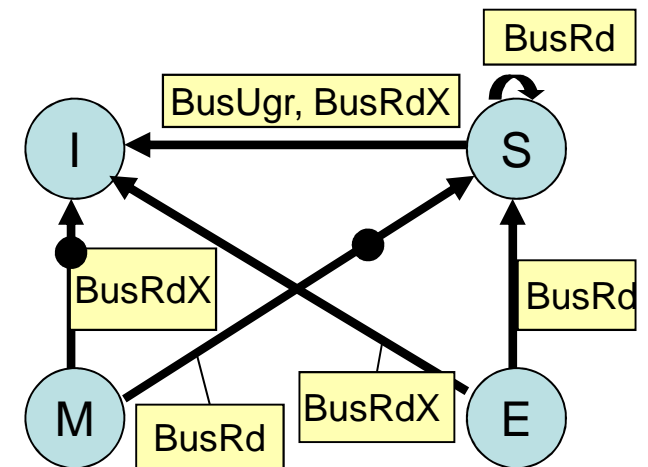


Příklad – Protokol MESI

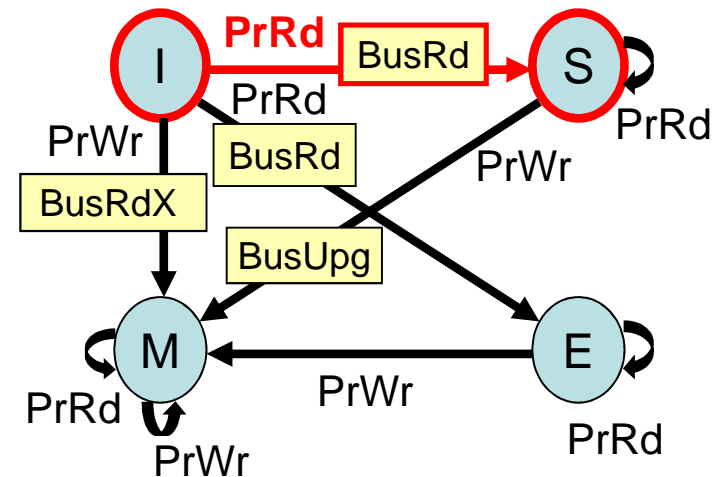
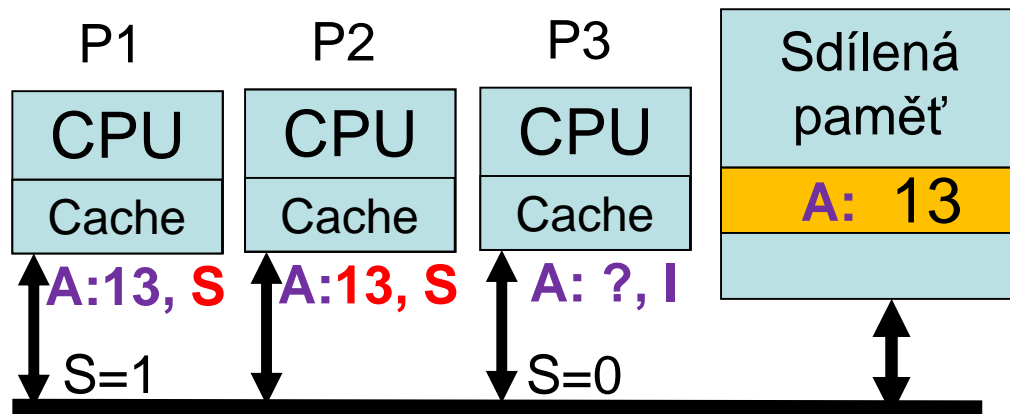


Procesor P1 chce načíst data z adresy A.

- P1 vyše PrRd(A) do cache, blok je ale Invalid, tzn. nastal read miss.
- Proto řadič cache musí provést čtení z paměti – vyše BusRd(A).
- Ani jeden slídící řadič cache neindikuje, že by měl kopii bloku ve své skryté paměti.
- Proto když paměť vystaví data, ty jsou přinesena a stav bloku u P1 projde po hraně I->E.

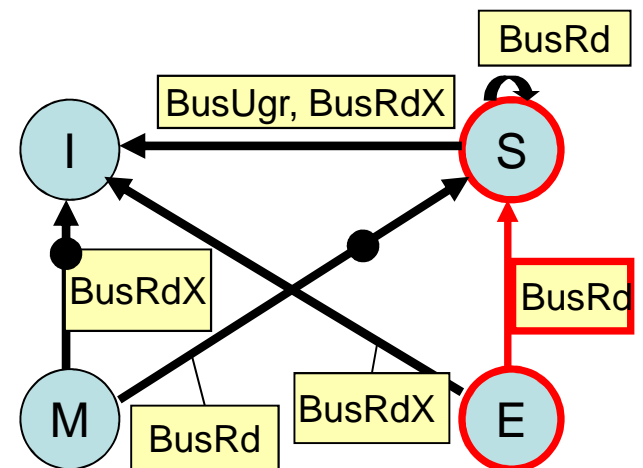


Příklad – Protokol MESI

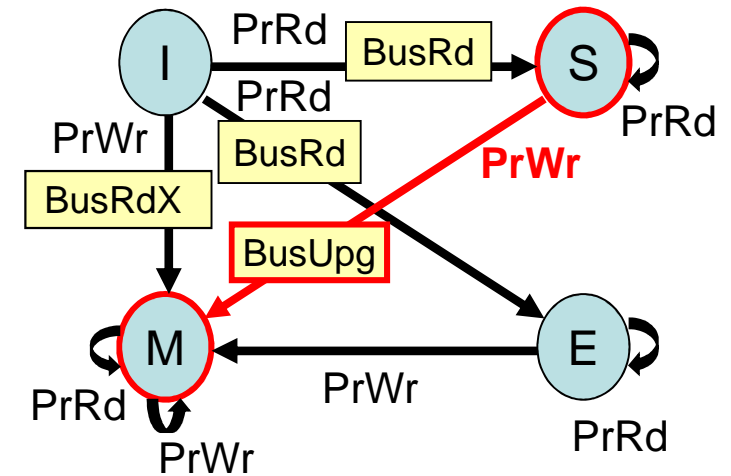
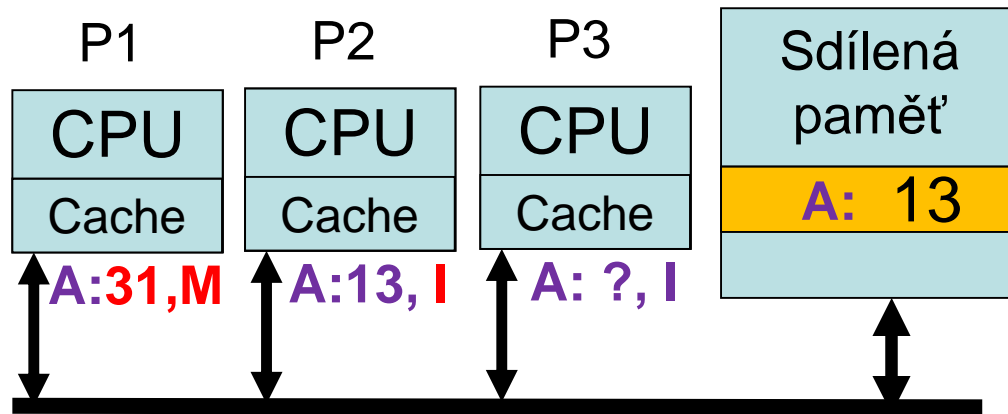


Procesor P2 chce načíst data z adresy A.

- P2 vyše PrRd(A) do cache, blok je ale Invalid, tzn. nastal read miss.
- Proto řadič cache musí provést čtení z paměti – vyše BusRd(A).
- Slídící řadič cache P1 indikuje, že má kopii bloku ve své skryté paměti, tzn. nastaví vodič S=1, zruší požadavek na čtení z paměti a vystaví data.
- Oba projdou po hraně do S. (P1: E->S; P2: I->S)

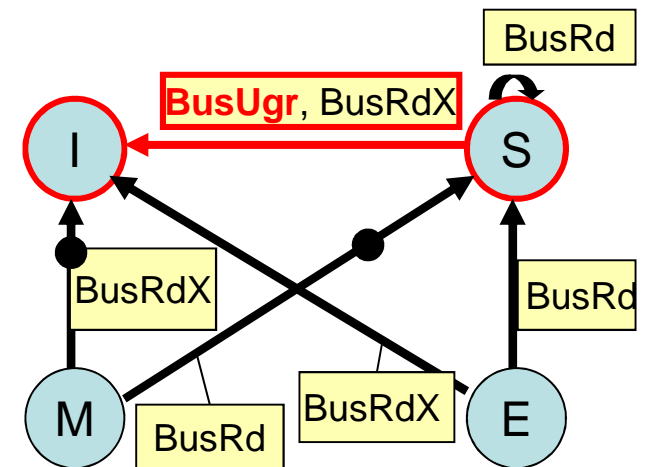


Příklad – Protokol MESI

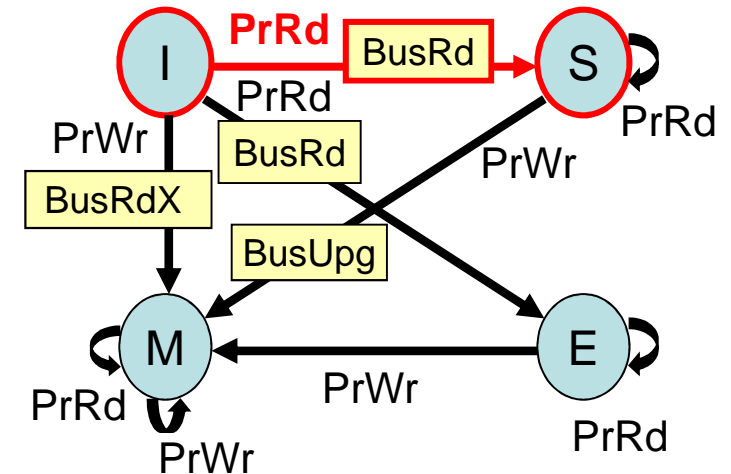
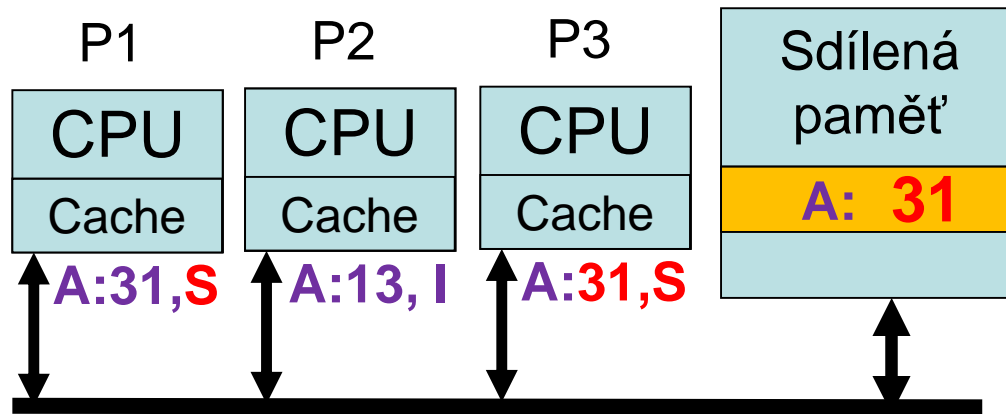


Procesor P1 zapíše novou hodnotu do A.
Například 31.

- P1 vyše PrWr do cache. Ta má blok ve stavu S.
- Proto řadič cache vyše BusUpgrade(A).
- Všechny slídící řadiče cache (v tomto případě jenom jeden) vidí BusUpgrade a projdou S->I.
- Paměť není aktualizována.
- Blok cache u P1 prošel po hraně S->M.

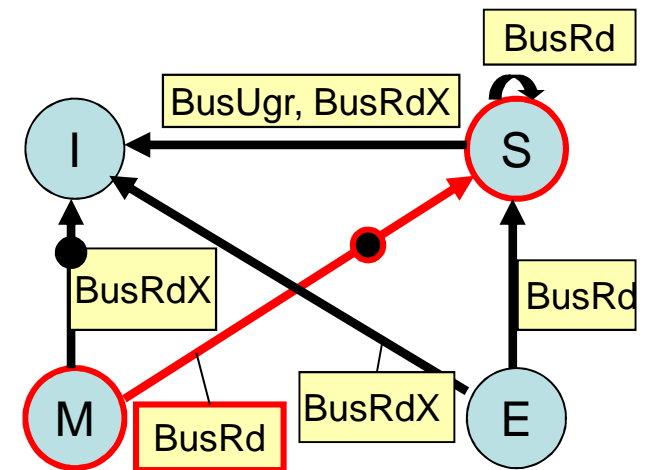


Příklad – Protokol MESI

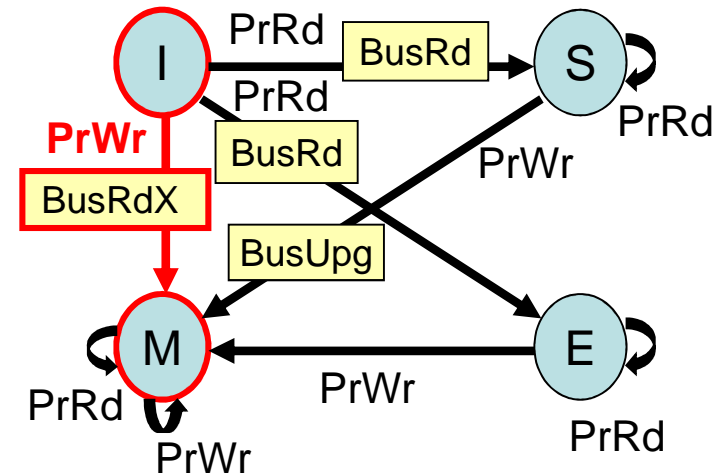
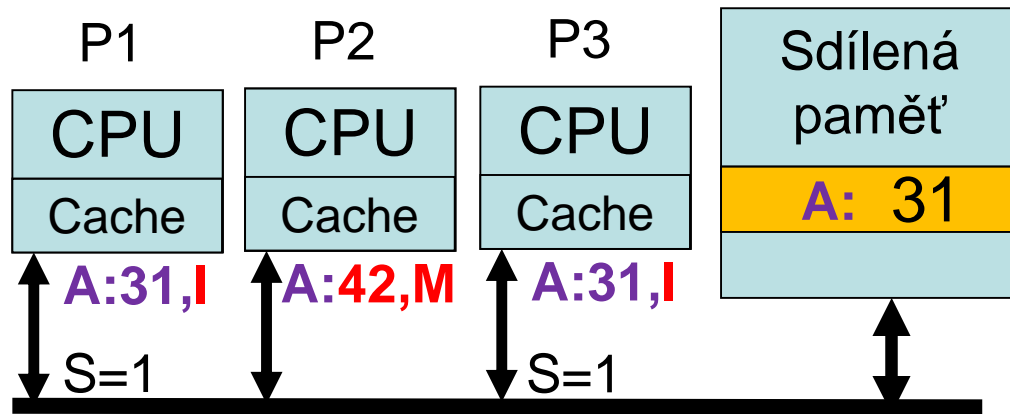


Procesor P3 chce načíst data z adresy A.
Jakou hodnotu načte?

- P3 vyšle PrRd(A) do cache, blok je ale Invalid. Proto řadič cache vyšle BusRd(A).
- Slídící řadič cache P1 indikuje, že má kopii bloku ve své skryté paměti, tzn. nastaví vodič S=1, zruší požadavek na čtení z paměti a vystaví data.
- Oba projdou po hraně do S. (P1: M->S; P3: I->S)
- Na hraně M->S je copy-back. Takže paměť je aktualizována.

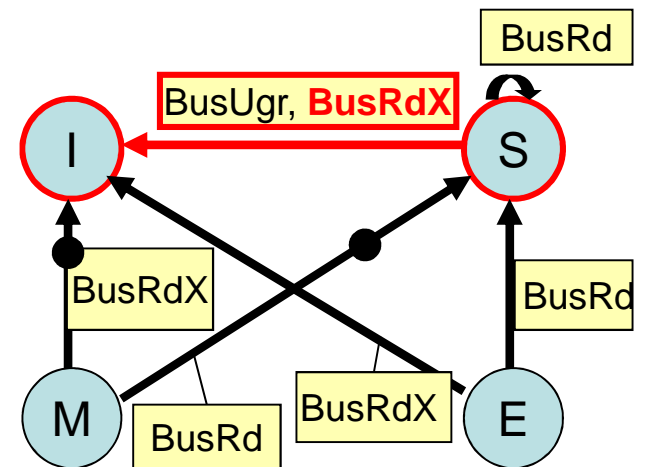


Příklad – Protokol MESI



Processor P2 chce zapsat data na adresu A.
Například 42.

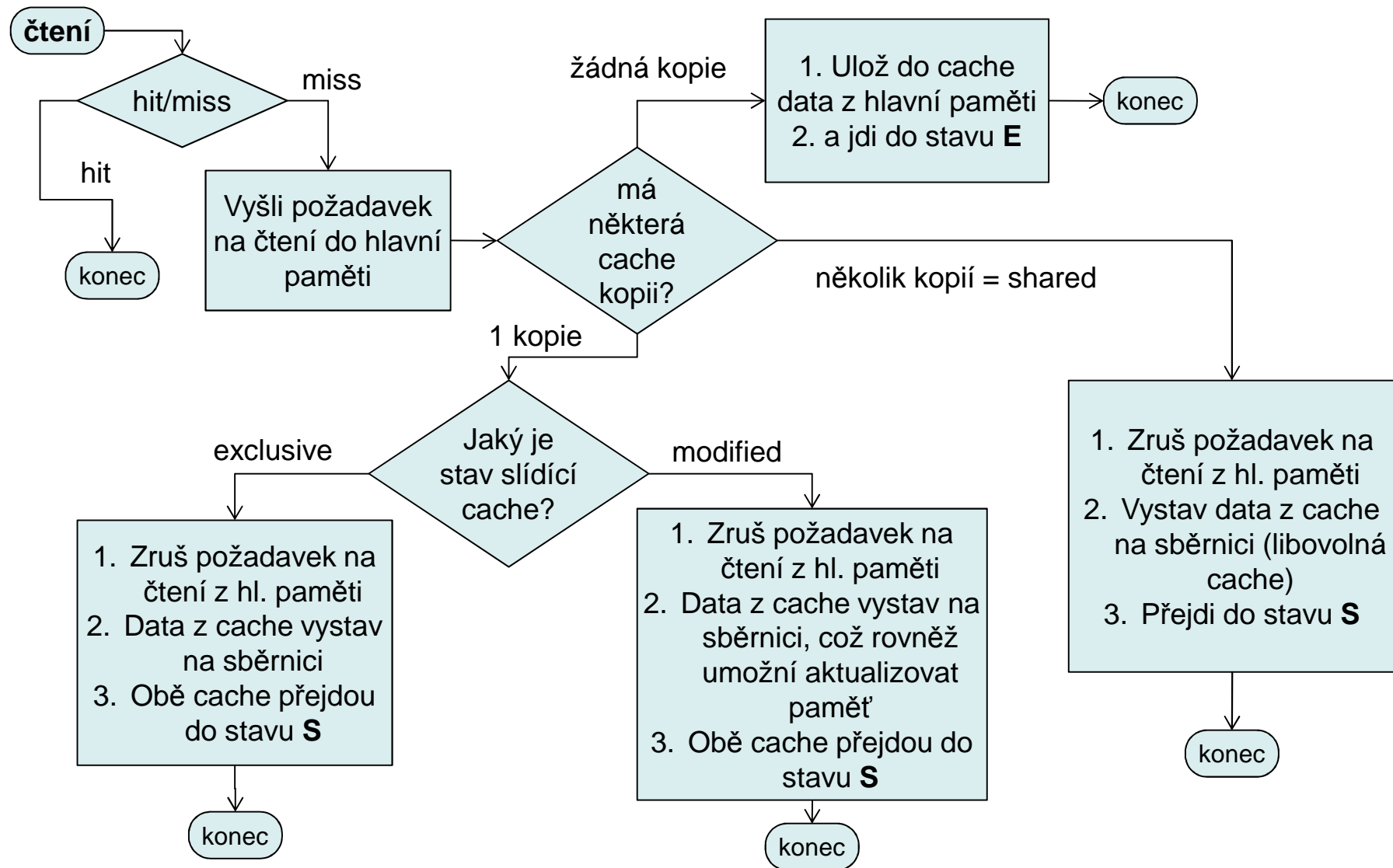
- P2 vyše PrWr(A) do cache, blok je ale Invalid. Proto řadič cache vyše BusRdX(A).
- Slídící řadiče cache indikují, že mají kopii bloku. Zruší se požadavek na čtení z paměti a některý z řadičů vystaví data.
- Oba slídící projdou po hraně do S->I.
- Žadající P2 projde po hraně I->M.
- Paměť není aktualizována.



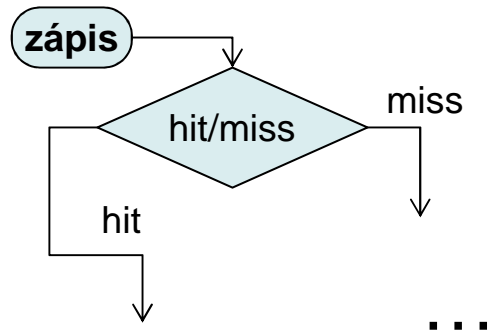
Poznámky ulehčující implementaci

- Znalost, že v cache uložená hodnota je E (Exclusive), vede na možnost nevysílat žádnou transakci.
- Změny stavu řádku v cache nastávají při události Zápis/Čtení (při přístupu do paměti).
- Událost způsobí buď
 - Aktivita připojeného procesoru (přístup do cache), nebo
 - Jako výsledek úspěšného slídění na sběrnici.
- Změny stavu řádku nastávají jen v případě shody adres.

Shrnutí předchozích slajdů.. Čtení

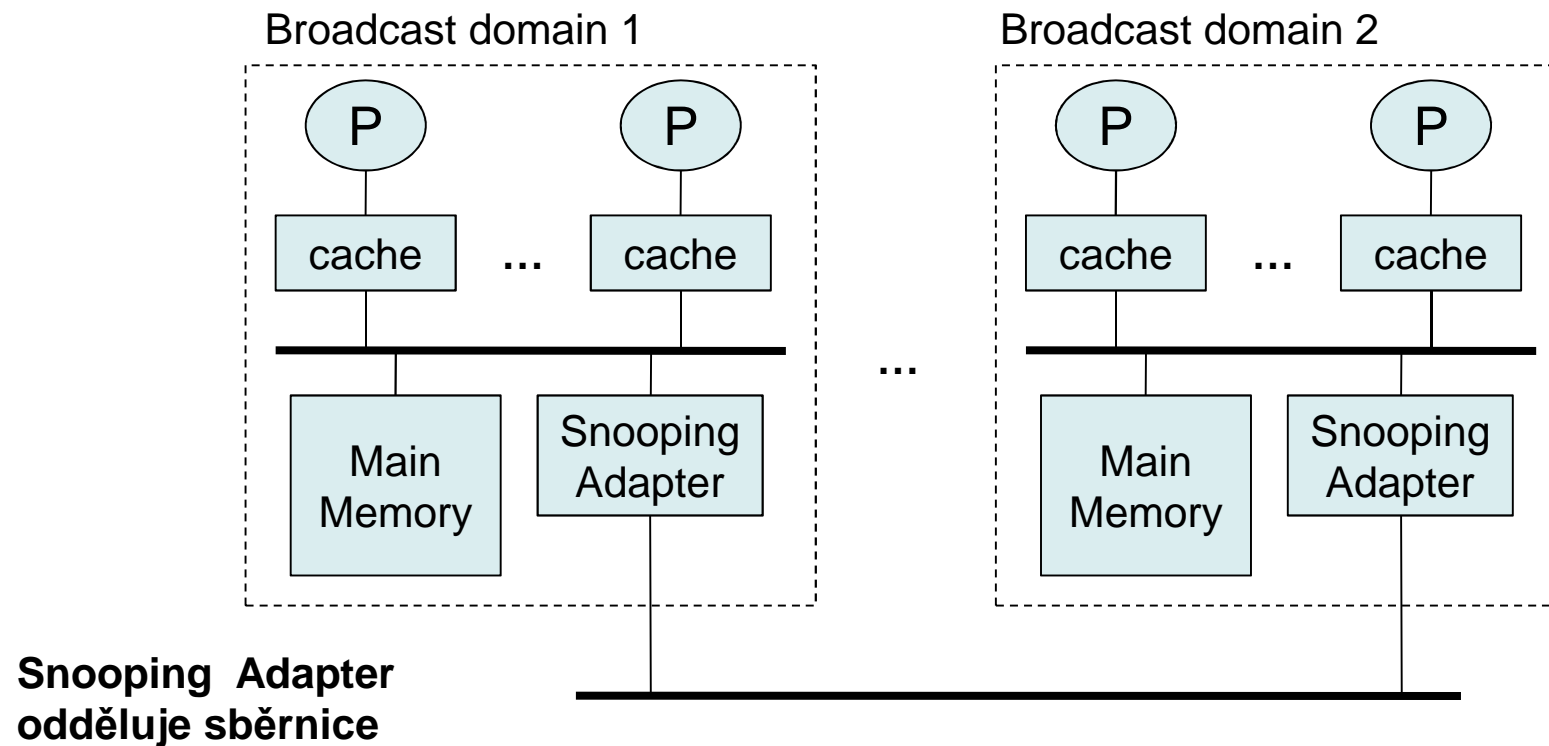


Shrnutí předchozích slajdů.. Zápis



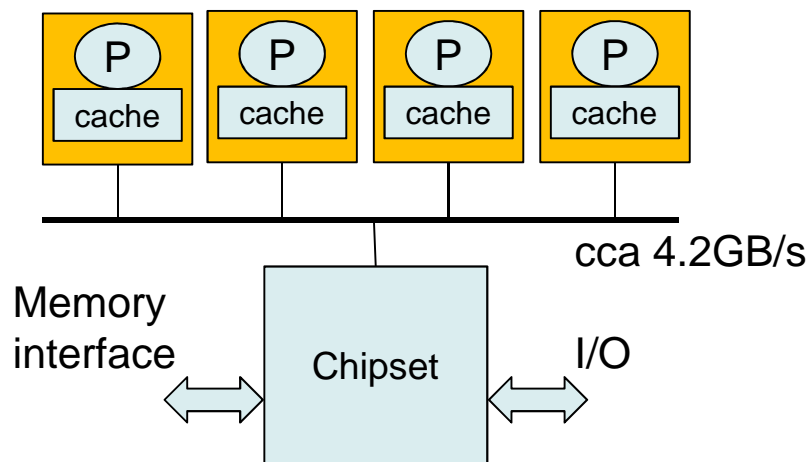
Rozšiřování (škálování) Broadcastu pro více procesorů

- Systémy založené na Broadcastu (slídění) lze reálně škálovat do cca 8-10 uzlů, kde každý uzel dnes může být vícejádrový procesor...
- Jednou z možností je hierarchické slídění (Hierarchical Snooping).
- Nicméně sběrnice jsou dnes již vytlačeny do ústraní a nahrazeny point-to-point propojením – viz další slajdy

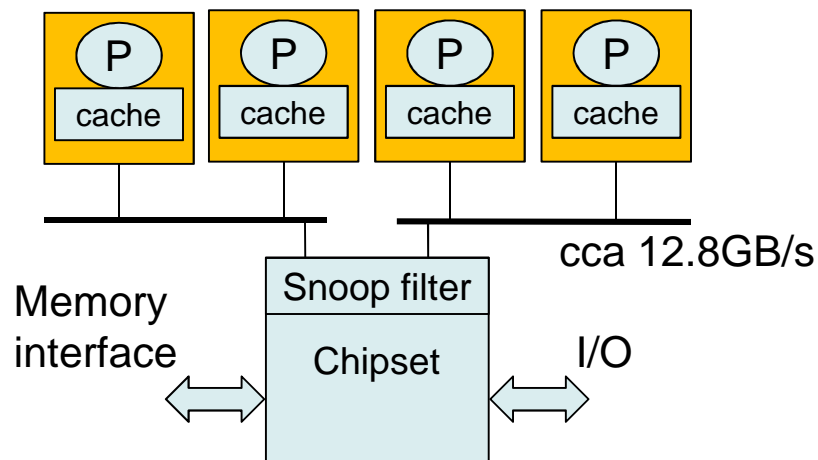


Pokud se podíváme na vývoj dovnitř procesoru...

Rok
2004



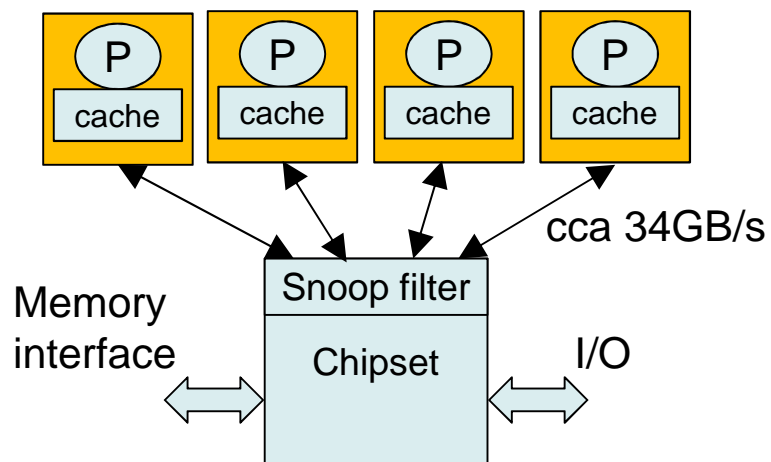
Rok
2005



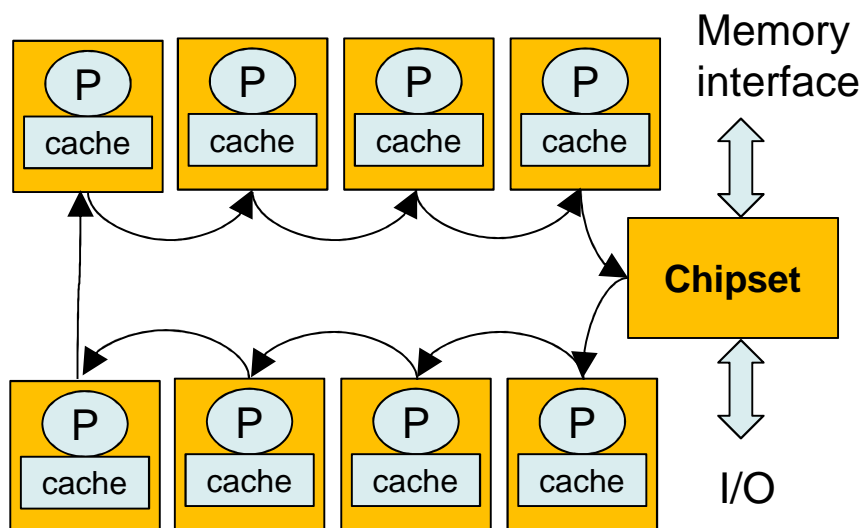
- Klasické omezení sběrnice.. Problém: zvyšování frekvence sběrnice. **Důležité:** Sběrnice poskytuje serializaci (v důsledku arbitrace) všech požadavků – žádné dva procesory nemůžou modifikovat tu samou cache line v tom samém čase
- Dvě nezávislé sběrnice – DIB (dual independent buses). Princip snoopování ale musíme zachovat. Pokud bychom propagovali veškerý provoz, degradujeme výkon. Proto snoop filter filtruje provoz tím, že si uchovává snoopovací informace a nepropaguje nerelevantní transakce

Pokud se podíváme na vývoj dovnitř procesoru...

Rok
2007

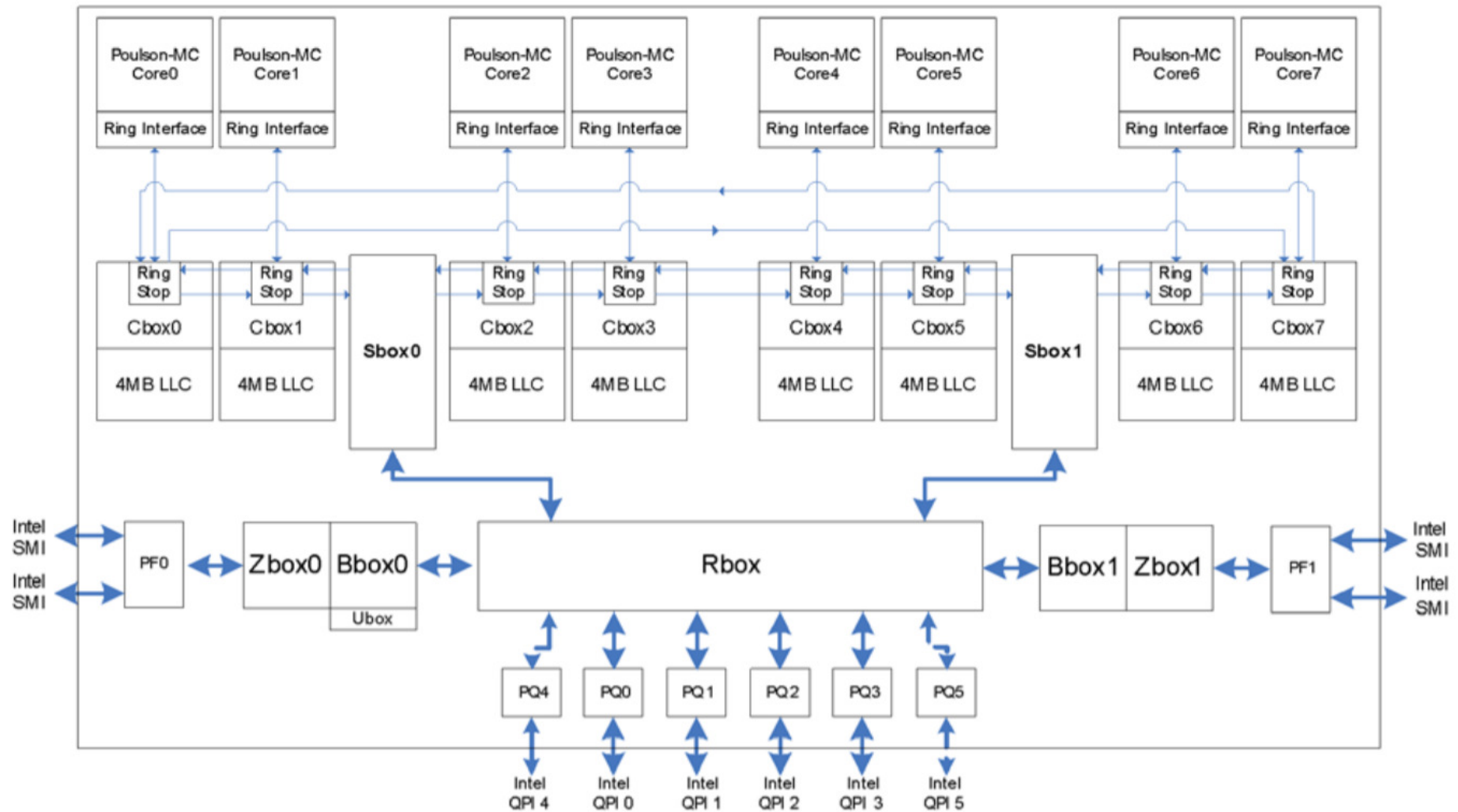


2009
až
nyní



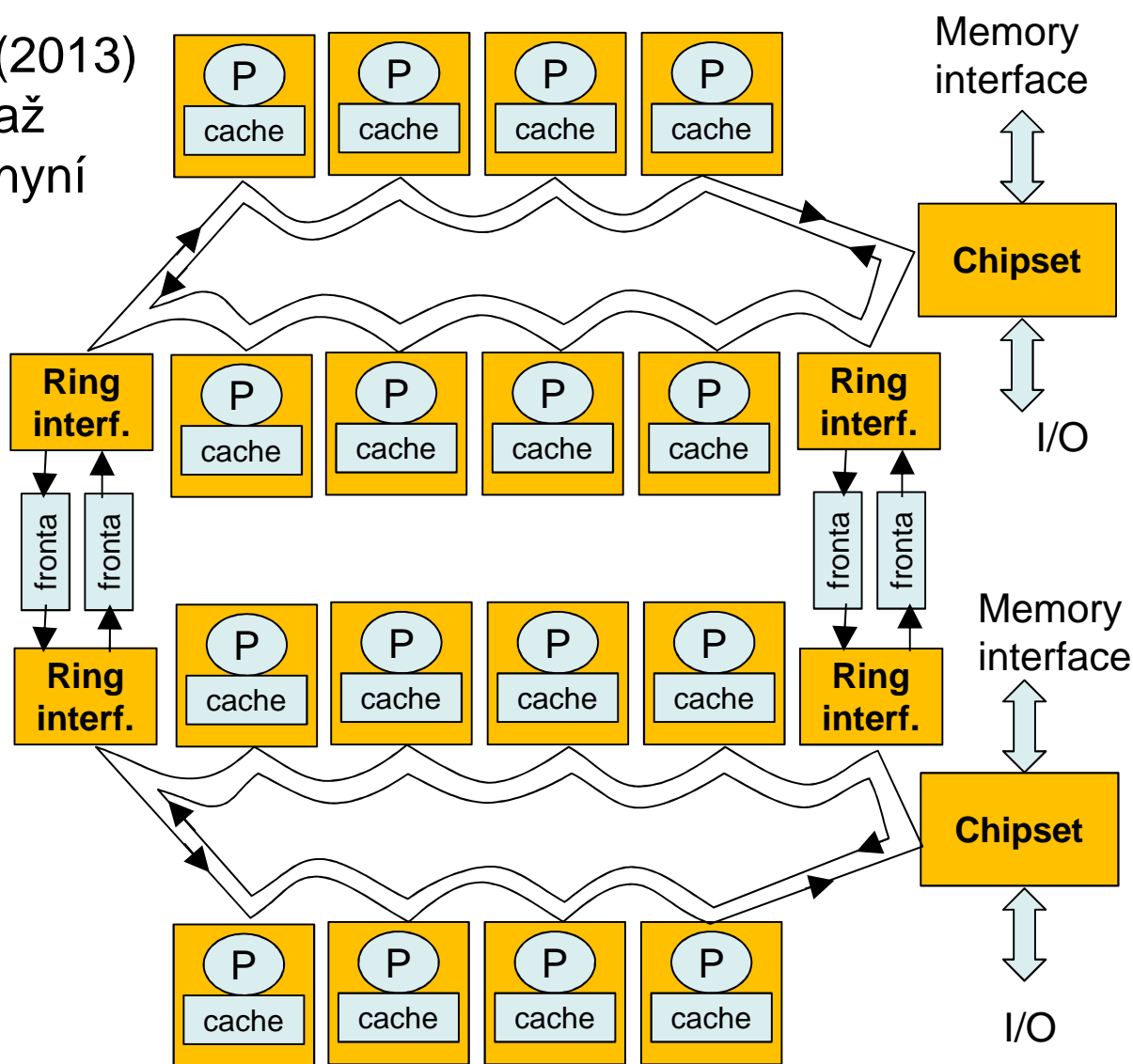
- Nástup point-to-point high-speed interconnects. Nicméně, snoop filter se stává úzkým hrdlem systému. Centralizace.
- Ring. Na obrázku jeden jednosměrný ring – všechny zprávy putují jedním směrem a mezi uzly nejsou nikdy přeuspořádány. Typicky dnes najdeme dva jednosměrné ringy v protisměru, tzv. bidirectional ring. Směrovače ringu mohou být velmi jednoduché – jako když jedete autem na kruhovou křižovatku a opustíte ji když potřebujete.

Příklad z praxe: Intel Itanium Processor 9500



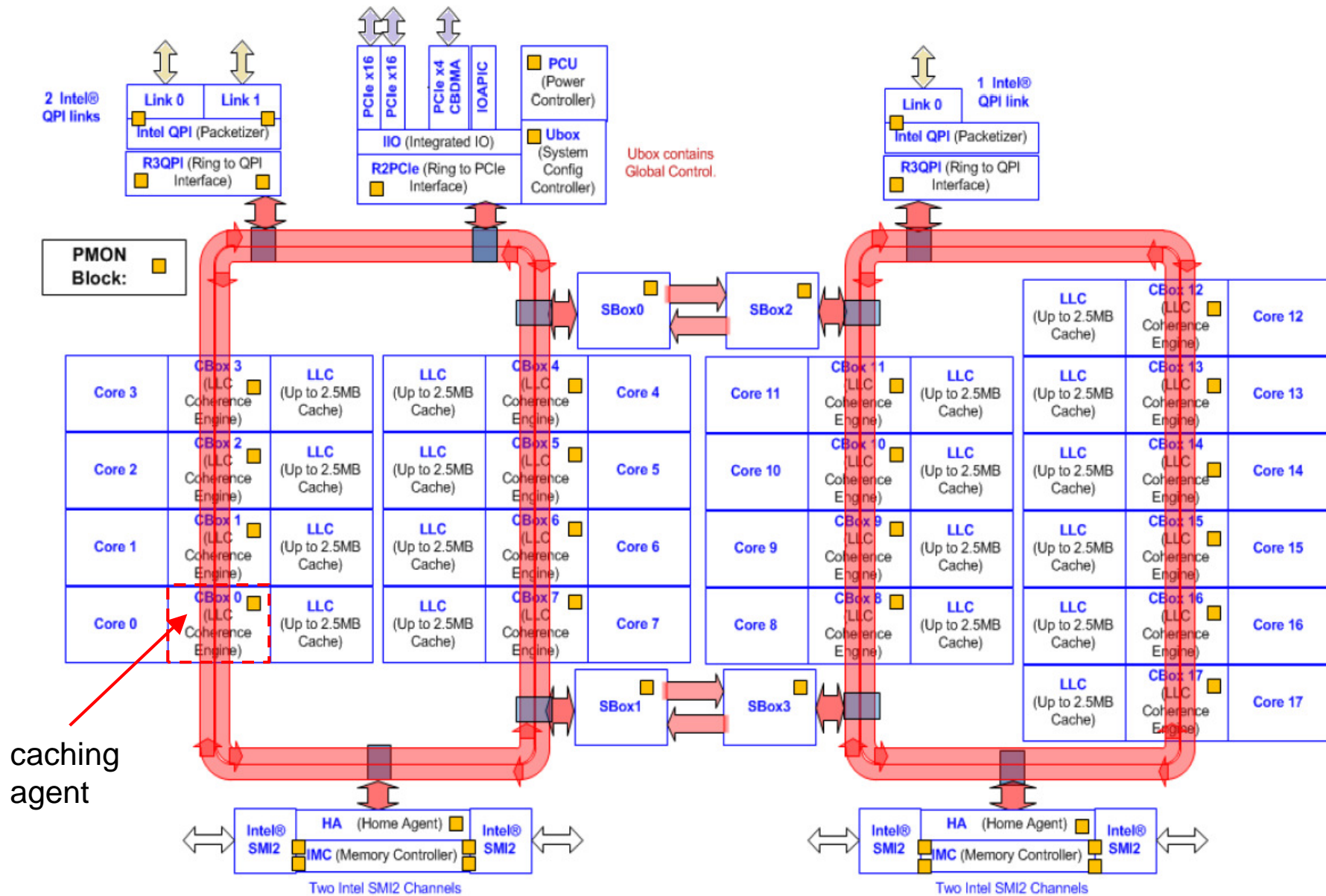
Pokud se podíváme na vývoj dovnitř procesoru...

(2013)
až
nyní



- Propojené ringy. Samotný procesor se stává systémem NUMA.
- Ring nabízí rychlé point-to-point propojení a zároveň odstraňuje složitost paketově-orientovaných propojení obecné topologie. Směrovač v každém uzlu obsahuje jeden vstupní a jeden výstupní port. Uzly mají možnost vložit nebo odstranit zprávu pomocí distribuované arbitrace. Nicméně ring neposkytuje seřazení zpráv jak tomu bylo v případě sběrnice – záleží na pozici pozorovatele... => Greedy snooping (IBM Power4/5), to ale znamená „všechno všem“, **nebo selektivně pomocí tzv. Directories.**

Příklad z praxe: Broadwell-EP (Intel Xeon)



Příklad z praxe: Broadwell-EP (Intel Xeon)

Broadwell-EP k urychlení „snoopování“ a snížení zátěže (komunikace) používá:

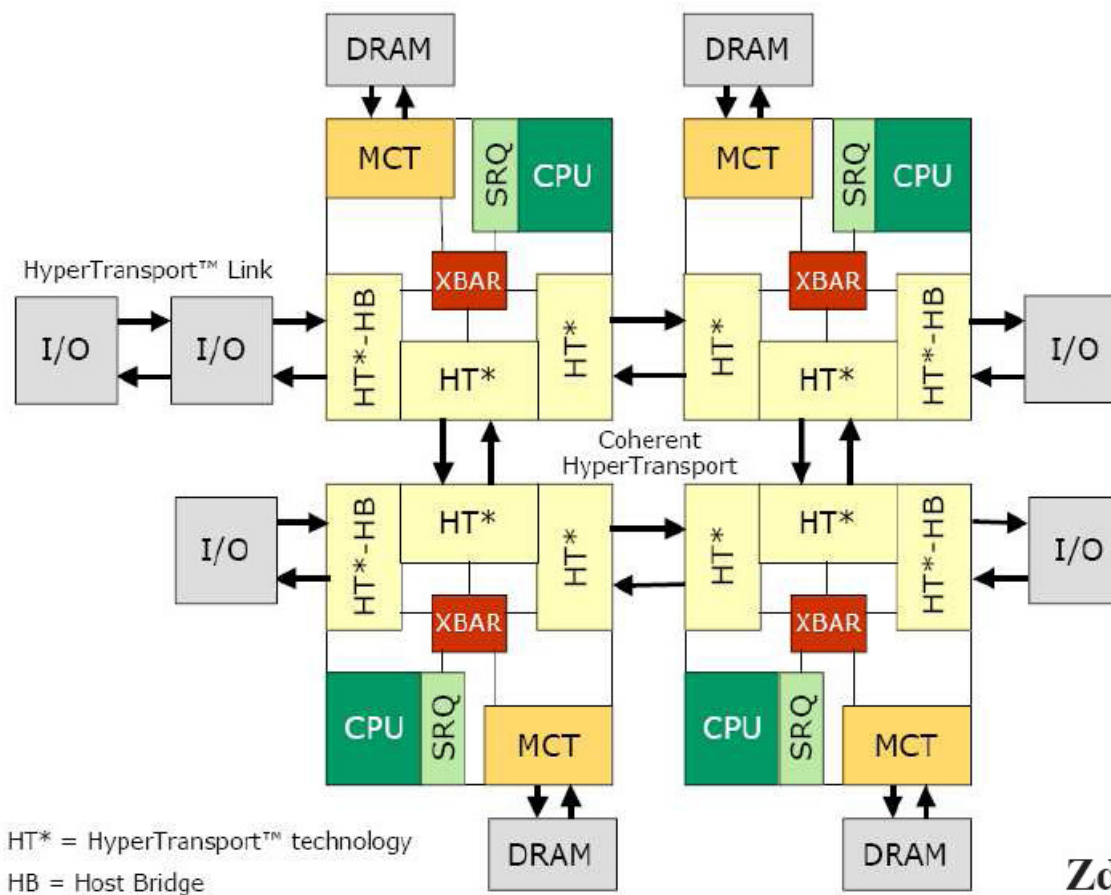
- **Directory Cache** – což je 14KB cache v každém HA (home agent). Ukládá 8-bit vektor, který udává, který CA (caching agent) může vydat kopii cache line. Directory cache se nachází na čipu. Directory cache hit znamená, že jsme zjistili koho se máme dál dotazovat po datech.
- **Directory** (adresář). Adresář se nachází v paměti a má pouze 2 bity (tzv. **directory bits**) na blok (cache line) - Local/Invalid, SnoopAll, Shared. Adresář se používá pouze v případě Directory cache miss.

Poznámky:

- **Directory cache** rozšíření k DAS protokolu. Urychluje přístup ke cache lines, které budou přeposílány z cache z jiných uzlů.
- **Directory assisted snoop broadcast protocol (DAS)** je rozšířením běžně používaného **MESI** protokolu (stav F znamená Forwarding – tj. kdo se stará o přeposílání). DAS používá tzv. adresář (directory) pro ukládání pomocných informací. Redukuje se tím počet snoopovacích dotazů z HA.

Jak se slídí, když není sběrnice?

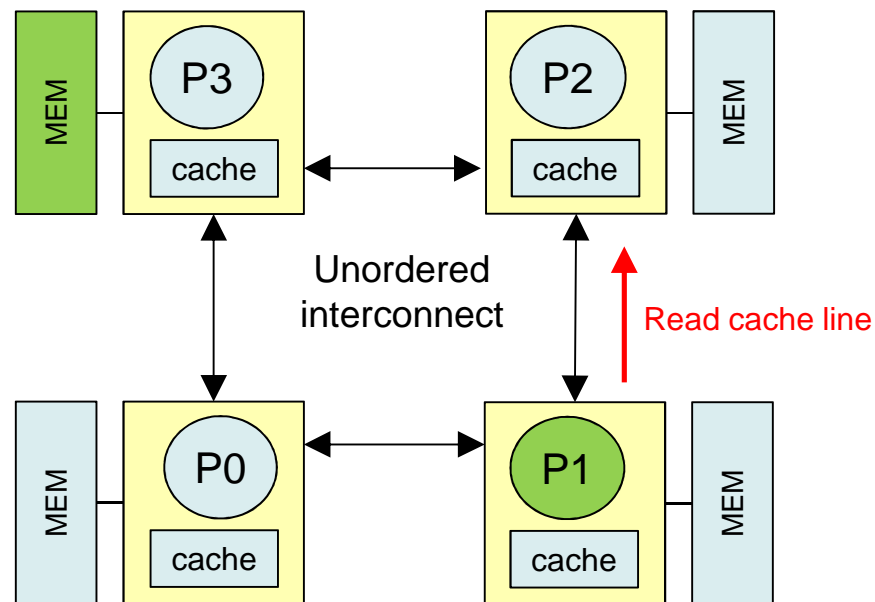
- Příklad: AMD Quad – Coherent HyperTransport
- Místo sběrnice najdeme HT (AMD) nebo QPI (Intel)
- Propojení **čtyř více-jádrových procesorů**:



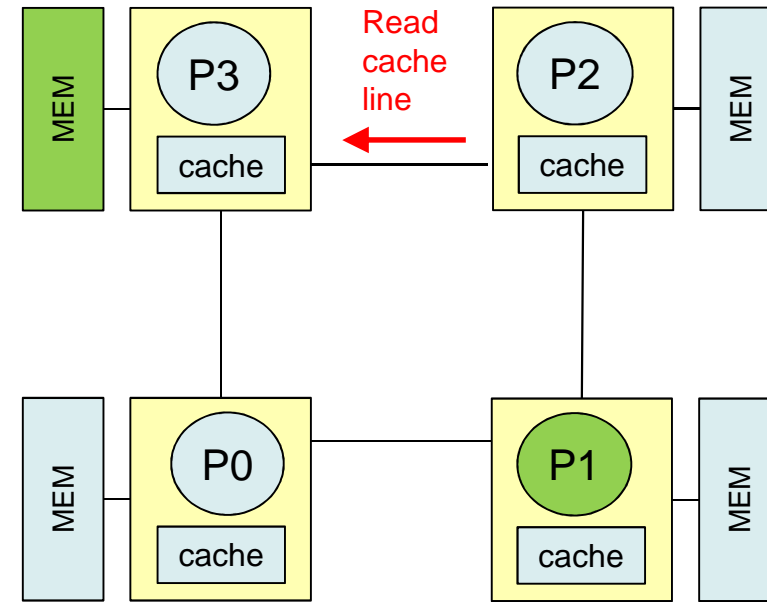
Jak se slídí, když není sběrnice? Broadcast protocol

Příklad: CPU1 čte položku, která je domovská v CPU3

Krok 1:



Krok 2:



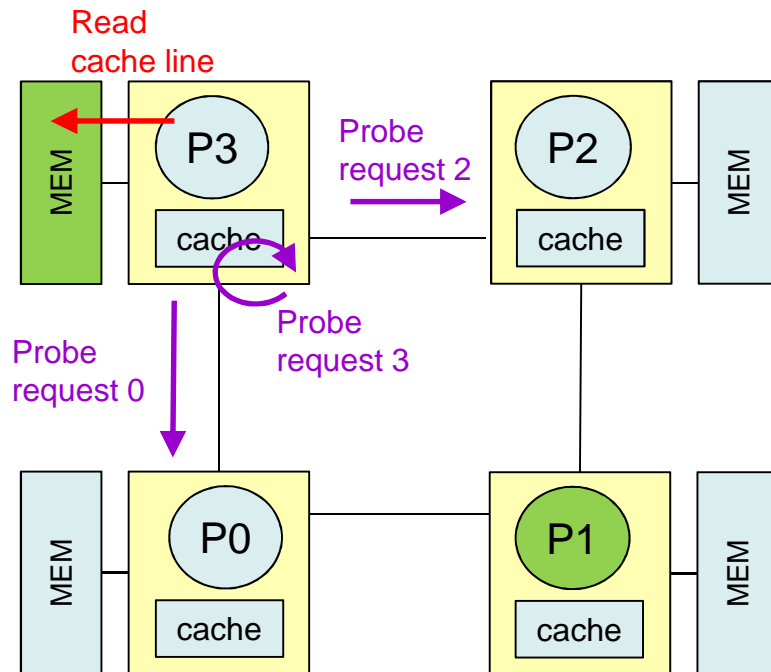
- Reakce na last-level cache miss: dotaz do home node
- Kde memory controller určí pořadí všech dotazů ke stejné cache line.

- Neexistuje přímé spojení P1 a P3. Proto se dotaz posílá přes P2

Jak se slídí, když není sběrnice? Broadcast protocol

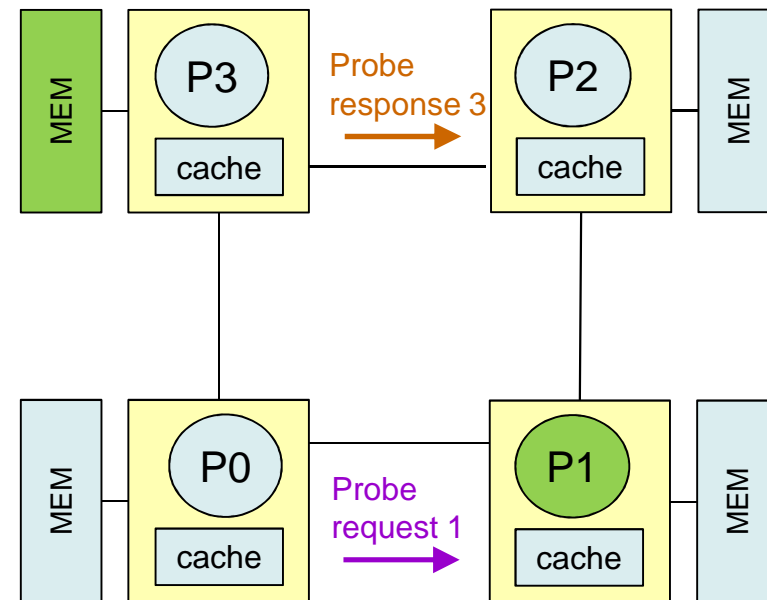
Příklad: CPU1 čte položku, která je domovská v CPU3

Krok 3:



- Když se dotaz v domovském uzlu stane aktivní, vyšlou se cache probe requests ke všem procesorům a zároveň se inicializuje přístup do RAM

Krok 4:

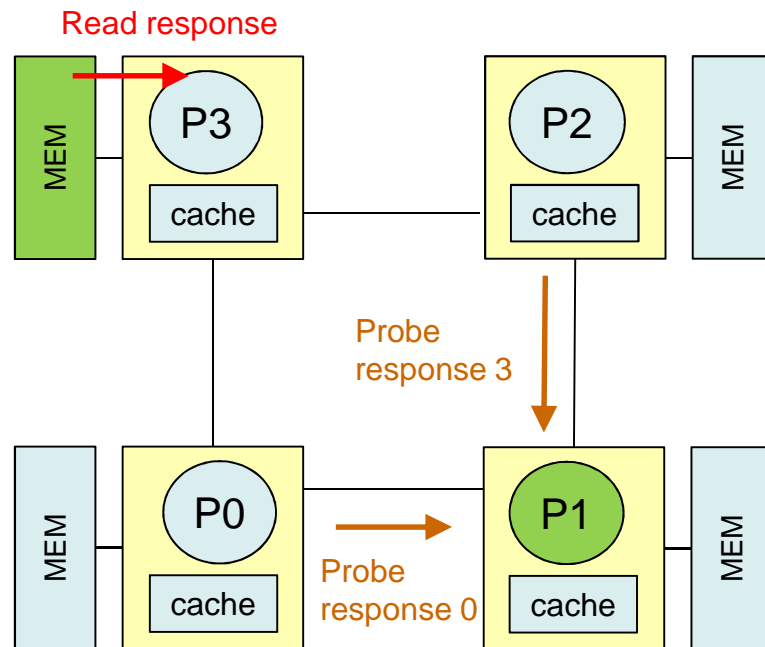


- Všechny procesory pošlou probe responses (hned jak je mají) k dotazujícímu procesoru – P1.
- Memory controller taky – až bude mít data z RAM – viz krok č.5.

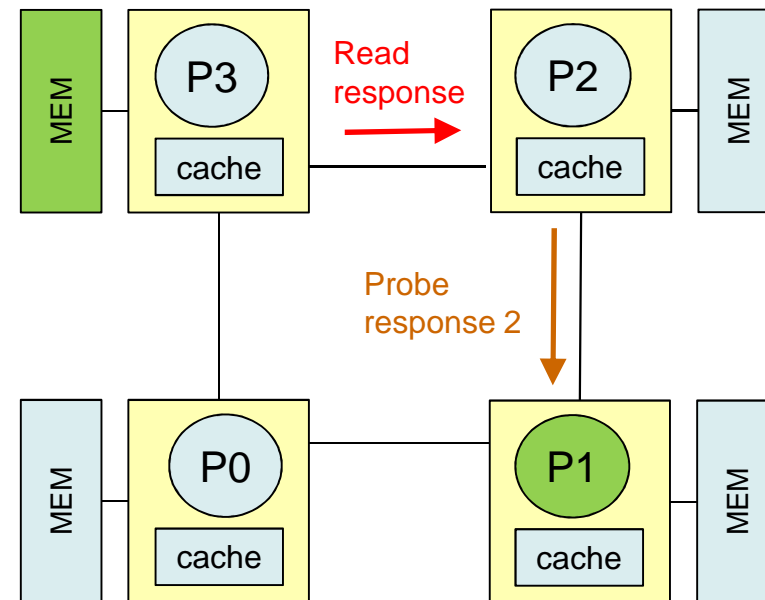
Jak se slídí, když není sběrnice? Broadcast protocol

Příklad: CPU1 čte položku, která je domovská v CPU3

Krok 5:



Krok 6:

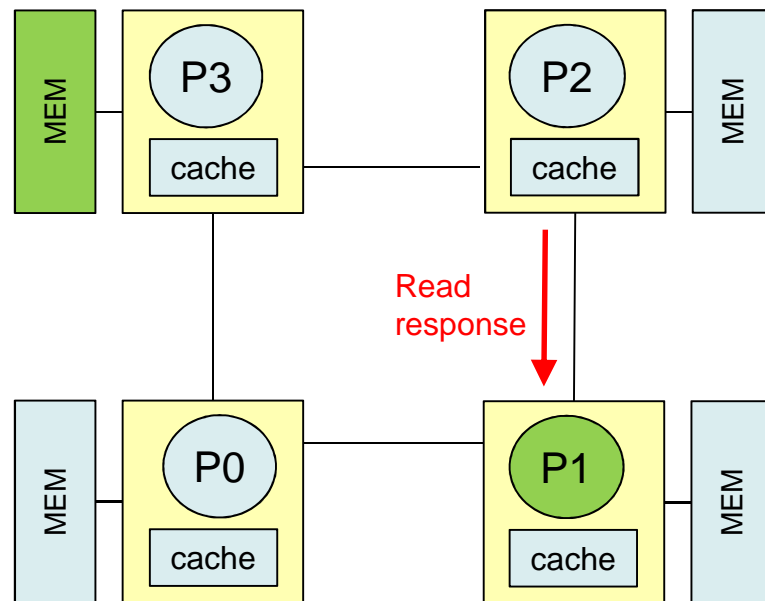


- Žadatel, procesor 1, postupně sbírá odezvy od všech procesorů

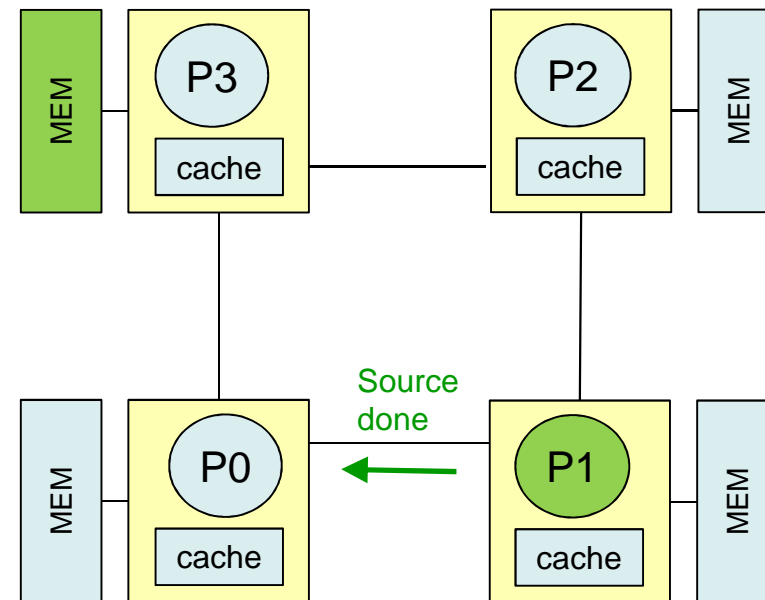
Jak se slídí, když není sběrnice? Broadcast protocol

Příklad: CPU1 čte položku, která je domovská v CPU3

Krok 7:



Krok 8:

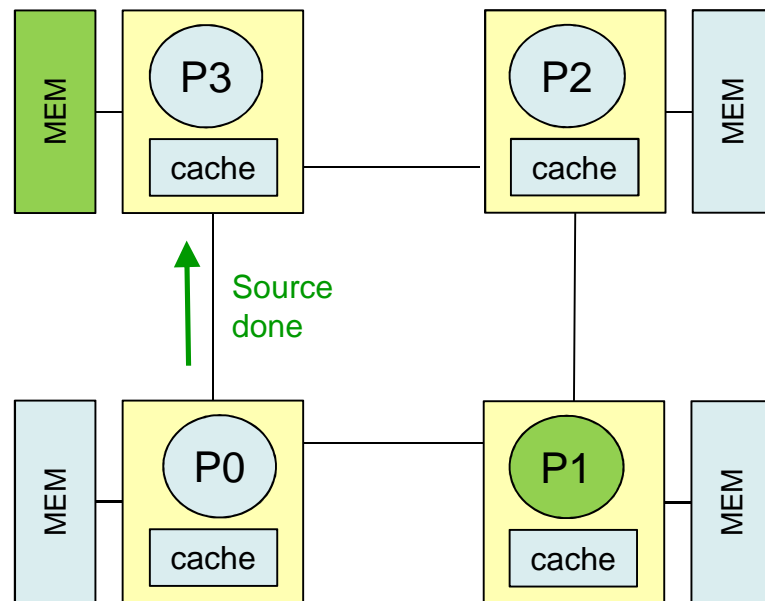


- Jakmile žadatel obdrží všechny responses, vyšle zprávu do home uzlu o tom, že cache line request byl vyřízen. To je signál řadiči paměti, že může začít vyřizovat jiný požadavek ke stejné cache line.

Jak se slídí, když není sběrnice? Broadcast protocol

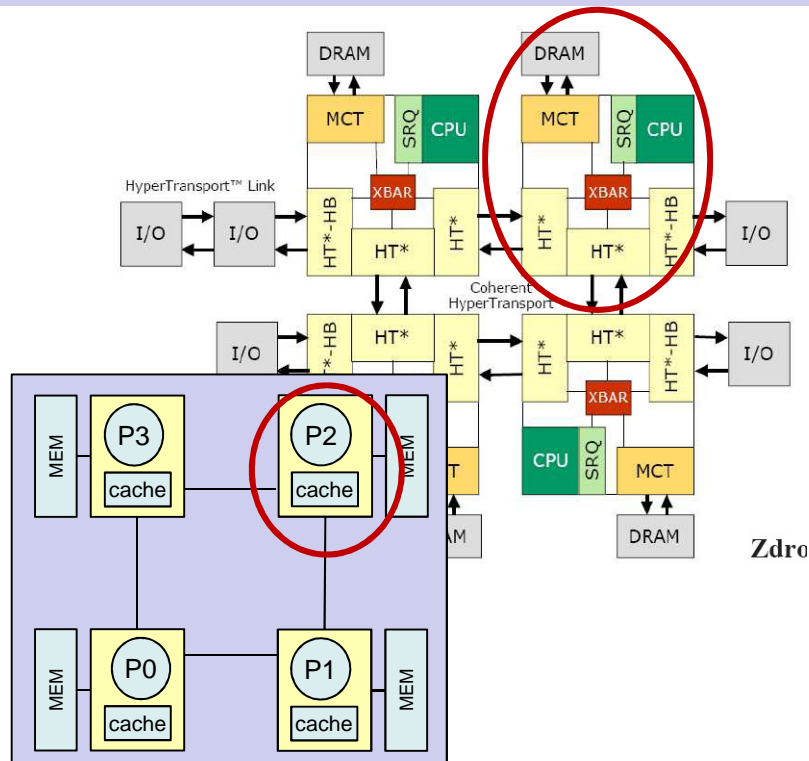
Příklad: CPU1 čte položku, která je domovská v CPU3

Krok 9:



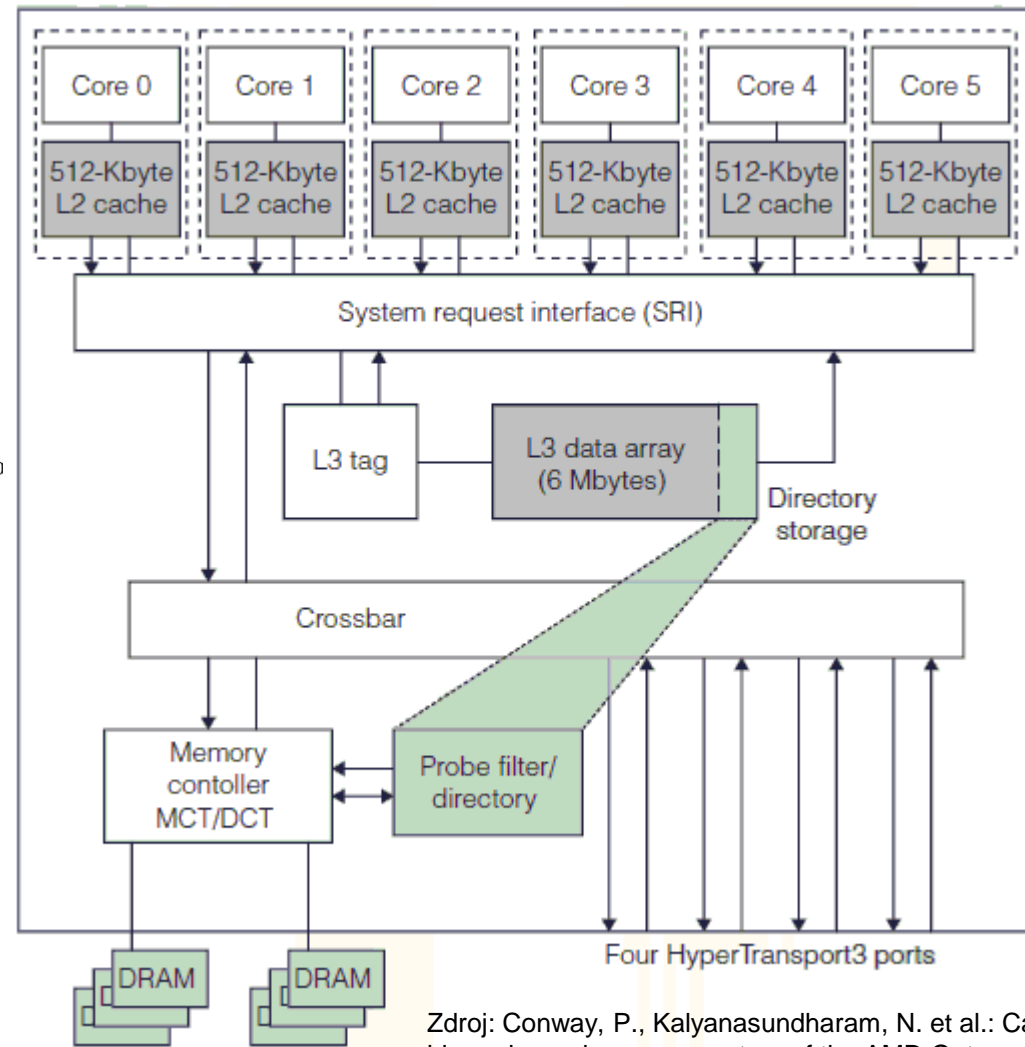
- Všimli jste si rozsah komunikace?
- Proto další generace procesorů od AMD již používala tzv. **HT assist** (HyperTransport Assist directory protocol) – cca od roku 2010
- HT assist používá **adresář**, který uchovává informaci o tom, které cache lines jsou cachovány v jiných CPU
- To umožňuje významně redukovat meziprocessorovou komunikaci. Místo původního broadcastu probes ke všem procesorům, nyní 3 možnosti:
- žádný probe – je to v RAM a nikdo nesdílí
- directed probe (dotaz pouze k jednomu CPU) – není to v RAM, ale má to jeden
- broadcast probe

Takhle vypadá AMD Quad – Coherent HyperTransport



Zdroj

- Probe Filter (HT Assist) - funguje tak, že používá část L3 cache jako directory cache v níž monitoruje již nakešované řádky. Místo generování množství žádostí (cache probes), procesor prohledá tuto část L3 cache.



Zdroj: Conway, P., Kalyanasundharam, N. et al.: Cache hierarchy and memory system of the AMD Opteron processor, IEE Micro, March/April 2010.

Pokud sběrnice nestačí nebo nechceme posílat všechno všem

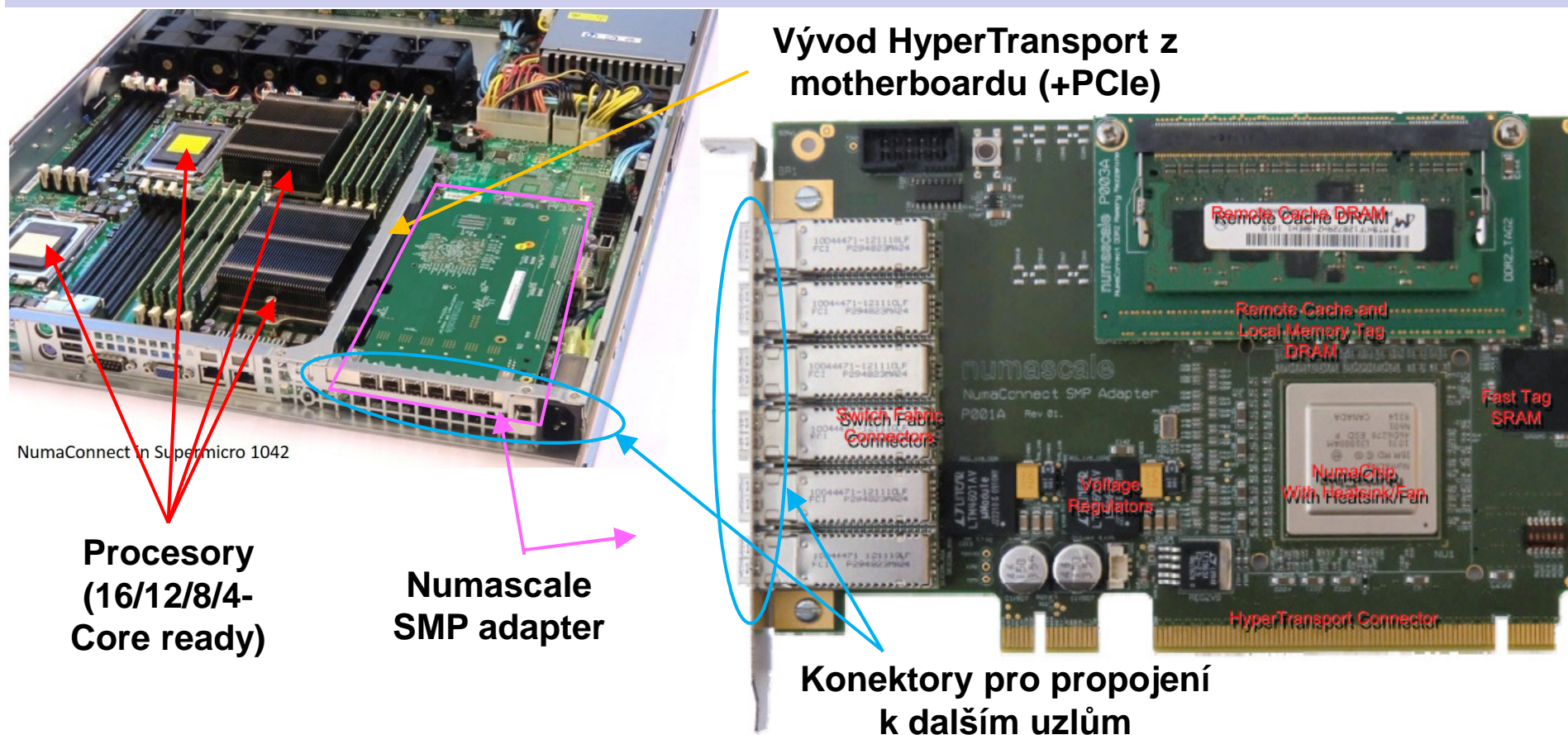
- Posílání informací všem ostatním (nebo poslouchání všemi) není škálovatelné řešení...
- Viděli jsme, že s odstraněním sběrnic (kdy odposlouchávání/snooping není problém), nástupem point-to-point propojení mezi jednotlivými jádry CPU (uzly) a nárustem počtu jader bylo potřeba vyřešit otázku jak odposlouchávat, ale zároveň nezatěžovat systém nadměrnou komunikací

- **Adresáře (Directories).**

(Koncem 80., začátkem 90. let vzniklo několik projektů se snahou vyřešit problém sdílené paměti. Jedním z nich byl “SCI” (Scalable Coherent Interface) - HP, Apple, Data General, Dolphin, US Navy,.. Dalším projektem byl “DASH” – Stanford. Oba mají jedno společné. Directory based cache coherence architecture.

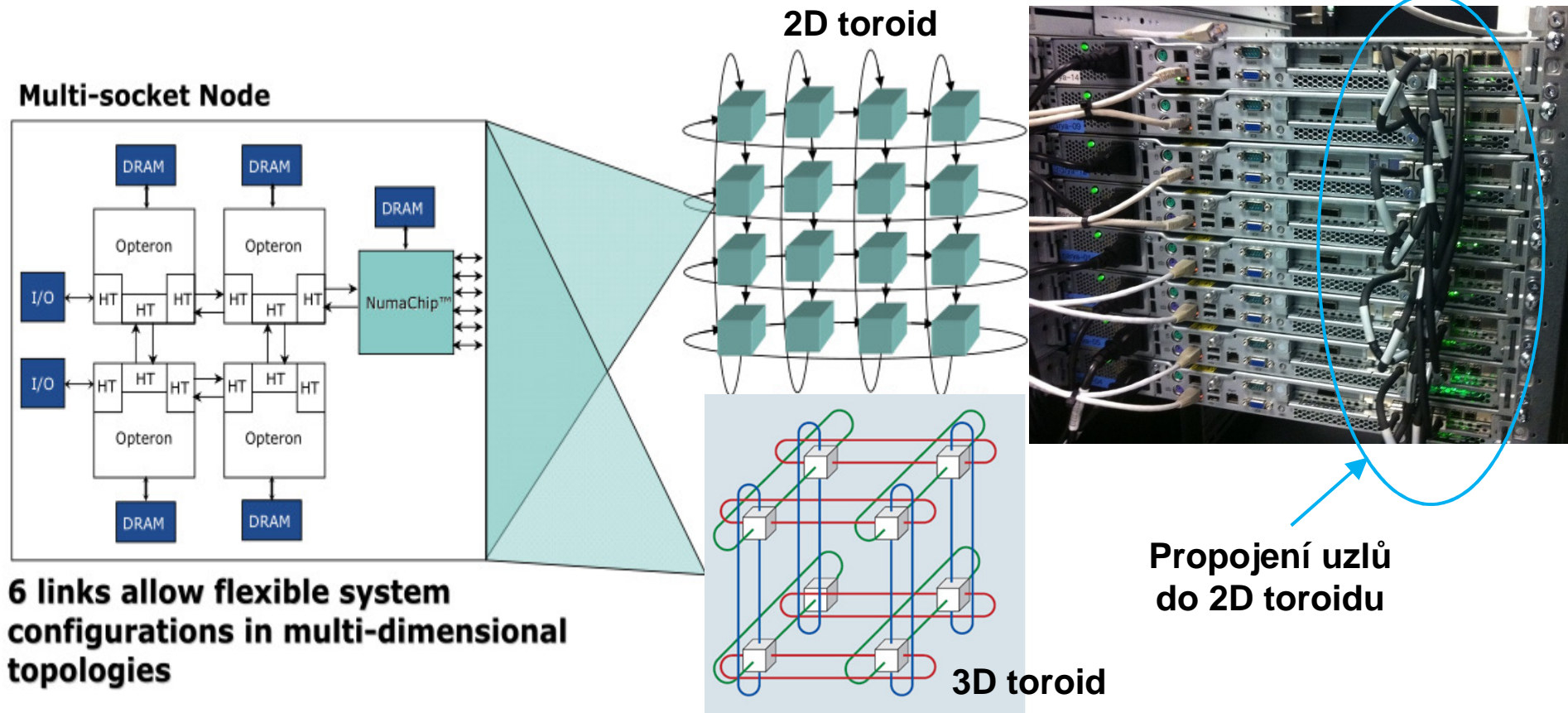
- Vhodné i pro propojení stovek nebo tisíců procesorů
- Používání Directories tedy není nová myšlenka. Jak jsme před chvílí viděli, pouze se Directories přesunuly i do dnešních běžných CPU

Příklad z praxe - Numascale



- Podpora cache koherence v HW (directory based)
- Max. 4096 uzlů (nodes). Jeden uzel: 4 procesory (obr.vlevo nahoře)
- uzel: 16 DIMM sockets x 32 GB = max. 512 GB
- Nicméně max. 256 TB celkem. Dáno omezením 48-bit fyzického adresního rozsahu CPU
- Například: 4GB Cache, 8 GB Tag (podporuje 240 GB Local Node RAM)

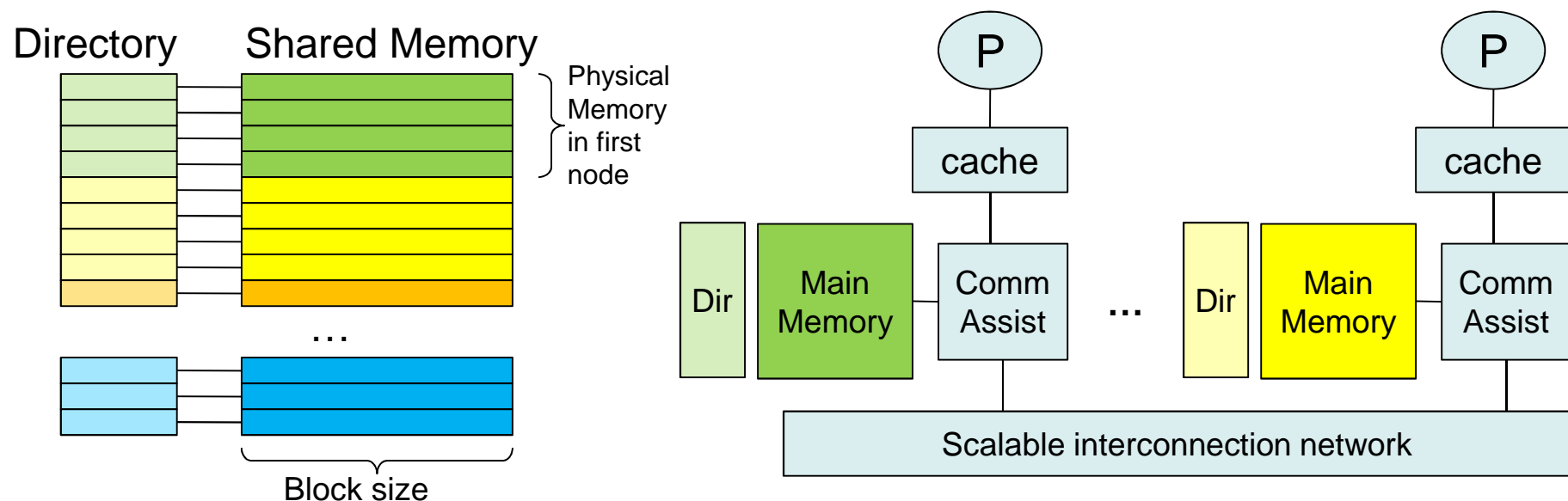
Příklad z praxe – Numascale – propojení uzlů



- 2D toroid (anuloid) – každý uzel 4 sousedy
- 3D toroid (anuloid) – každý uzel 6 sousedů
- Max. 4096 uzlů x 4 procesory/uzel = 16 384 více-jádrových procesorů
- Program, který dokáže běžet na jednom uzlu, dokáže běžet i na celém systému bez změny. (OpenMP, MPI, Threads)

Directories – začněme od začátku...

- Pokud broadcast (multicast) nelze snadno realizovat (sběrnice)
- Základní myšlenka: Mít adresář (Directory), který indikuje pro každý řádek/blok paměti:
 - zda je v cache (alespoň v jedné)
 - ve které cache/ích se nachází
 - zda je v cache dirty nebo clean



Directories

- **Full directory** obsahuje kompletní informace pro každý řádek paměti. Pro n procesorový systém, Boolovský vektor délky $n+1$. Pokud bit i ($i=1,2,\dots,n$) je nastaven, i -ta cache má kopii řádky z paměti. Bit 0 pak indikuje zda je clean nebo dirty (v tom případě jenom jeden další bit může být nastaven = řádek je jenom v jedné cache)
- V NUMA systému, každý uzel má jenom část adresáře obsahující informace o řádcích uložených v jeho paměti = **home node**, ostatní: **remote node**
- Při cache miss, požadavek je poslán do domovského uzlu
- Full directory – nevýhodou je velikost adresáře.
Například pro 8 procesorový systém kde velikost L3 cache řádků je 64 B (předpokládáme, že koherence je aplikována na úrovni L3) bude velikost adresáře 2% ($9/(64*8)=0.018$) velikosti sdílené paměti, ale pro 64 procesorů 13% ($65/(64*8)=0.127$) .

Příklad realizace Full directory

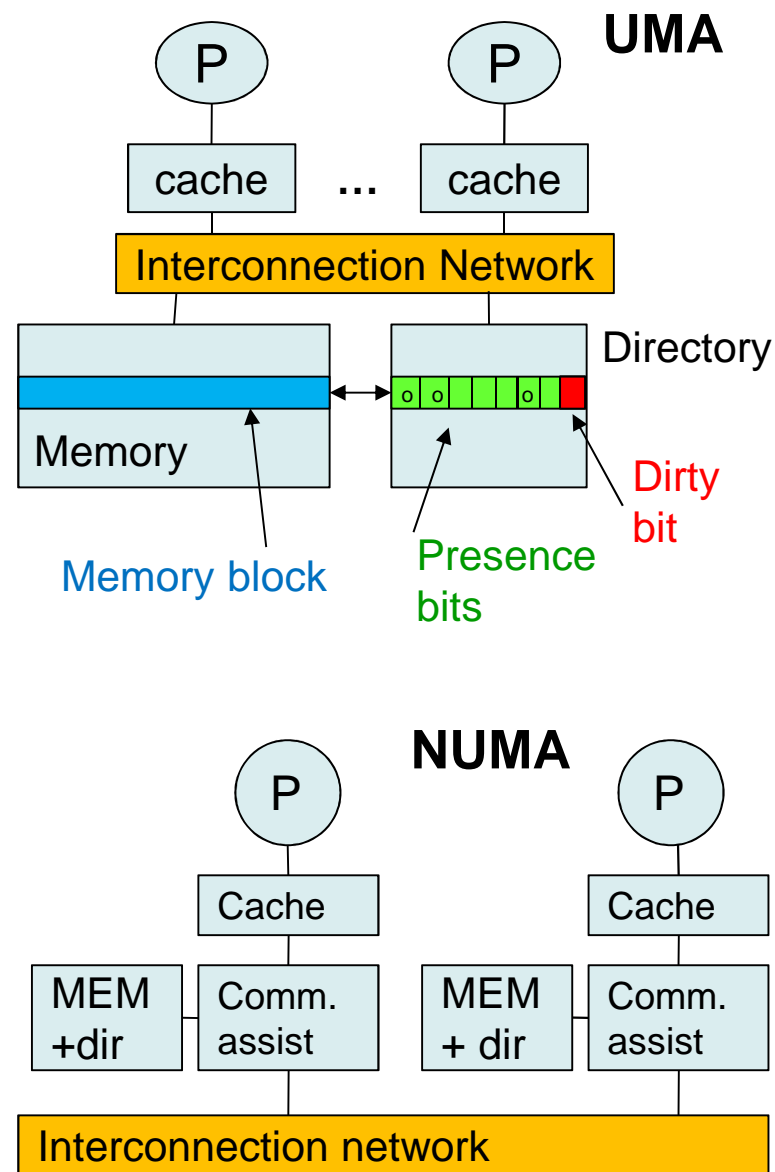
- Mějme K procesorů. Pro každý blok v paměti máme 1 Dirty-bit, K Presence-bits.

Čtení bloku procesorem „i“ (po read miss):

- Je-li dirty-bit OFF potom { čti z hlavní paměti; nastav p[i] ON; }
- Je-li dirty-bit ON potom { získej blok od dirty procesoru, update paměti; nastav dirty-bit OFF; nastav p[i] ON; pošli data k i; }

Zápis do paměti procesorem „i“ (po write miss):

- Je-li dirty-bit OFF potom { pošli invalidace ke všem sdíleným kopiím; pošli data do i, vynuluj všechna p[j] v adresáři a pouze pro p[i] nastav ON; dirty bit ON; }
- Je-li dirty-bit ON potom {získej blok od dirty procesoru; původní dirty procesor: invalid; p[i] nastav ON jenom pro nový dirty node}
- Poznámka 1: Když je dirty bit ON, jenom jeden node (dirty node) může cachovat daný blok a tedy jenom jeden presence bit je ON
- Poznámka 2: Každý Cache block v cache má: MESI, MOESI, nebo jiný stav.



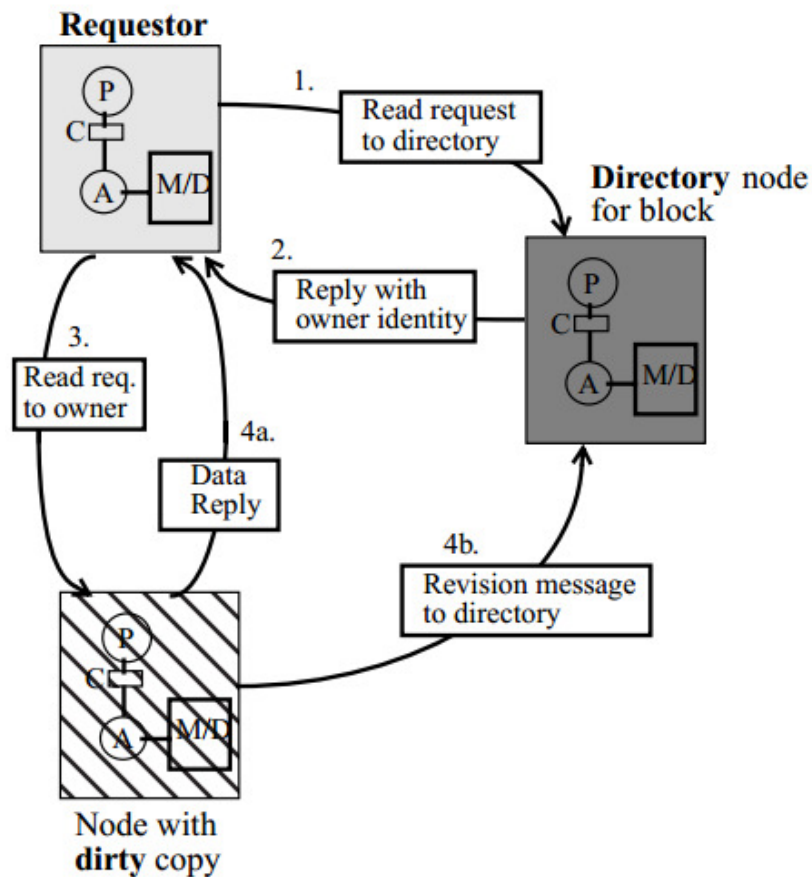
Directories – úvodní konstatování

- Řešením koherence pro rozsáhlé systémy CC-NUMA jsou adresáře (Directories) – jde o tzv. Directory Based CC-NUMA (Cache-Coherent NUMA).
- Adresář i paměť jsou distribuovány.
- Příklad SGI Origin2000 – 512 uzlů x 2 procesory = 1024 x MIPS R10K
- Myšlenka je založena na tom, že máme informaci o tom, kde jsou kopie každého jeho bloku a v jakém jsou stavu. Tato informace je obsažena v adresáři (directory). To umožňuje nepoužívat Broadcasty, ale omezené množství dvoubodových transakcí.
- **Domovský uzel** (home node) je ten uzel, kterého paměť obsahuje inicializovaná data, ostatní uzly jsou **vzdálené** (remote node).
- **Počet sdílených kopií bývá malý i v rozsáhlých systémech, proto je úspora oproti Broadcastu značná.**
- Cenou je existence další servisní datové struktury – adresáře (Directory): 4GB uzel, blok 128B, 256 procesorů => Velikost adresáře 32 Mx 8b (bit-mapa) = 32 MB.

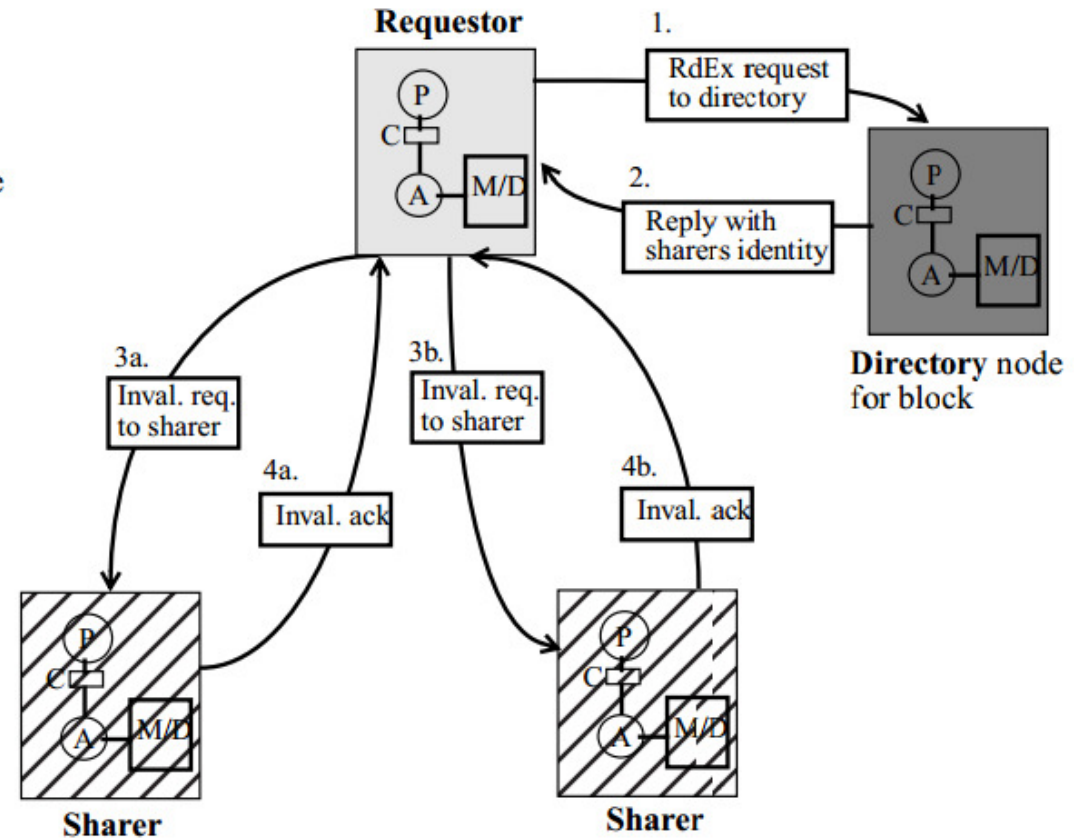
Vzhledem k danému bloku paměti nebo cache:

- **Home node** – domovský uzel: uzel, kterému patří alokovaný blok paměti – tam, kde byl alokován
- **Dirty node** – „špinavý“ uzel: uzel, který drží kopii daného bloku paměti ve své cache a je ve stavu modifikován
- **Owner node** – vlastník: uzel, který drží platnou kopii daného bloku paměti a zodpovídá za dodání dat ve chvíli kdy jsou potřeba (může to být home nebo dirty node)
- **Local node** – lokální uzel (**requestor** – žádající): ten, který se dotazuje po daném bloku paměti
- **Remote node** – vzdálený uzel: vzhledem k local node jsou to ty ostatní

Podrobněji



(a) Read miss to a block in dirty state

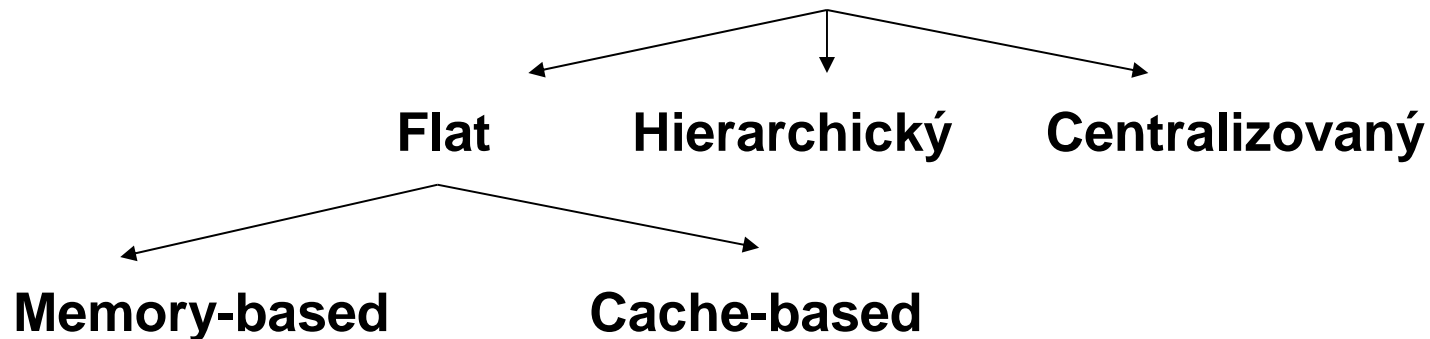


(b) Write miss to a block with two sharers

Jak zjistíme, kde se nachází patřičná část adresáře pro daný blok paměti?

- Nejčastější možností je tzv. **Flat Directory**, kdy ona část adresáře se nachází na fixně daném místě – obvykle v domovském uzlu, a ten poznáme podle adresy bloku (adresy, ze/do které chceme číst/zapisovat).
- Jinou možností je hierarchický adresář (hierarchical directory), kdy paměť je rozložena mezi uzly jako obvykle, ale adresář je uchováván ve formě stromu (logická struktura). Uzly (procesory) se nacházejí v listech stromu a uzly stromu uchovávají informaci pro daný blok: zda jeho potomci mají kopii bloku nebo ne. Reakcí na miss je pak procházení stromu směrem k rodičům. V praxi jsou pak uzly stromu distribuovány mezi uzly systému (procesory), takže při miss-u se obvykle generuje několik transakcí mezi uzly systému než se dohledá požadovaná informace.
- Další možností je centralizovaný adresář – jedno místo pro dotazování – přichází v úvahu pouze pro malé systémy – například udržování koherence uvnitř více-jádrového procesoru

Directory Storage Schemes



- Flat directory - **Memory based** – dirty uzel nebo všichni sdílející jsou identifikováni v domovském uzlu (záznam v paměti). Příkladem je již diskutovaný full-direktory. Alternativně je možno ukládat pouze čísla procesorů, které mají kopii (funguje pro malý počet sdílejících – což je obvyklá situace). Systém ovšem musí umět řešit případ, kdy je počet sdílených bloků větší (například vynucené zneplatnění bloku ve vzdálené cache ...)
- Flat directory - **Cache based** – záznam v home node již neobsahuje informace o všech sdílejících, ale pouze pointer na prvního sdílejícího (plus stavové informace). Informace o dalších sdílejících jsou uloženy v distribuovaném obousměrném seznamu. Další sdílející se tedy dohledají prohledáním seznamu. Cache, která obsahuje kopii bloku paměti, rovněž uchovává pointer na předchozího a následujícího sdílejícího.

Sumarizace a závěr

- Základní technikou pro zajištění koherence je slídění – protokol MESI, nebo častěji MESIF
- Místo slídění na sběrnici se dnes posílají slídící dotazy po point-to-point propojení – typicky ring, nebo 2D mesh
- U většího počtu uzlů se používají Directories
- Hybridní technologie – snoopy/directory systems
- Velkým problémem zůstává programátorský model pro víceprocesorové systémy a složitost ladění paralelních programů.
- Rozsáhlé mnohprocesorové systémy typicky zabezpečují koherenci paměti v rámci výpočetního uzlu (několik více-jádrových CPU) – OpenMP. Výměna dat mezi uzly je v režii programátora – MPI (Message Passing Interface). => Kombinace OpenMP + MPI.

Použité zdroje:

1. Shen, J.P., Lipasti, M.H.: Modern Processor Design : Fundamentals of Superscalar Processors, First Edition, New York, McGraw-Hill Inc., 2005
2. Bečvář M: Přednášky Pokročilé architektury počítačů.
3. <https://www.cs.utexas.edu/~pingali/CS395T/2009fa/lectures/mesi.pdf>
4. D.E.Culler, J.P. Singh, A.Gupta: Parallel Computer Architecture: A HW/SW Approach, Morgan Kaufmann Publishers, 1998.
5. Einar Rustad: Numascale. Coherent HyperTransport Enables the Return of the SMP
6. https://www.numascale.com/numa_pdfs/numaconnect-white-paper.pdf
7. Rajesh Kota: HORUS: Large Scale SMP using AMD Opteron processors. Newisys Inc., a Sanmina-SCI Company. http://www.hypertransport.org/docs/tech/horus_external_white_paper_final.pdf
8. <http://www.intel.com/content/dam/doc/white-paper/quick-path-interconnect-introduction-paper.pdf>
9. David Culler, Jaswinder Pal Singh, Anoop Gupta: Parallel Computer Architecture. A Hardware / Software Approach.
10. <http://www.intel.com/content/dam/www/public/us/en/documents/datasheets/itanium-9300-9500-datasheet.pdf>
11. https://tu-dresden.de/zih/forschung/ressourcen/dateien/abgeschlossene-projekte/benchit/2015_ICPP_authors_version.pdf?lang=de
12. Michael R. Marty: Cache Coherence Techniques for Multicore Processors, 2008.
13. <http://www.intel.com/content/dam/www/public/us/en/documents/datasheets/itanium-9300-9500-datasheet.pdf>